UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Ingeniería del Software e Inteligencia Artificial

TESIS DOCTORAL

Monitorización y descubrimiento para la gestión de redes auto-organizativas en redes virtualizadas y definidas por software

Monitoring and discovery for self-organized network management in virtualized and software defined networks

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Ángel Leonardo Valdivieso Caraguay

Director

Luis Javier García Villalba

Madrid, 2018

Monitorización y Descubrimiento para la Gestión de Redes Auto-Organizativas en Redes Virtualizadas y Definidas por Software

Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks

Thesis by

Ángel Leonardo Valdivieso Caraguay

In Partial Fulfillment of the Requirements for the Degree of
Doctor por la Universidad Complutense de Madrid en el
Programa de Doctorado en Ingeniería Informática

Advisor

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, April 2017

Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks

Thesis by

Ángel Leonardo Valdivieso Caraguay

In Partial Fulfillment of the Requirements for the Degree of
Doctor por la Universidad Complutense de Madrid en el
Programa de Doctorado en Ingeniería Informática

Advisor

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, April 2017

Monitorización y Descubrimiento para la Gestión de Redes Auto-Organizativas en Redes Virtualizadas y Definidas por Software

TESIS DOCTORAL

Memoria presentada para obtener el título de
Doctor por la Universidad Complutense de Madrid
en el Programa de Doctorado en Ingeniería Informática

Ángel Leonardo Valdivieso Caraguay

Director

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Abril de 2017

Dissertation submitted by Ángel Leonardo Valdivieso Caraguay to the *Departamento de Ingeniería del Software e Inteligencia Artificial* of the *Universidad Complutense de Madrid* in Partial Fulfillment of the Requirements for the Degree of *Doctor por la Universidad Complutense de Madrid en el Programa de Doctorado en Ingeniería Informática*.

Madrid, 2017.

(Submitted April 24, 2017)

Title:

Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks

PhD Student:

Ángel Leonardo Valdivieso Caraguay (angevald@ucm.es)
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain

Advisor:

Luis Javier García Villalba (javiergv@fdi.ucm.es)

This work has been done within the Group of Analysis, Security and Systems (GASS, http://gass.ucm.es/), Research Group 910623 from the Universidad Complutense de Madrid (UCM) as part of the activities of the research project funded by the European Commission Horizon 2020 Programme under Grant Agreement number H2020-ICT-2014-2/671672-SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks). This research has also been supported by Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT - Convocatoria Abierta 2012 (Quito, Ecuador). Part of this work was done during my stay in Portugal at University of Aveiro (Institute of Telecommunications).

Tesis Doctoral presentada por el doctorando Ángel Leonardo Valdivieso Caraguay en el Departamento de Ingeniería del Software e Inteligencia Artificial de la Universidad Complutense de Madrid para la obtención del título de Doctor por la Universidad Complutense de Madrid en el Programa de Doctorado en Ingeniería Informática.

Terminada en Madrid el 24 de Abril de 2017.

Título:

Monitorización y Descubrimiento para la Gestión de Redes Auto-Organizativas en Redes Virtualizadas y Definidas por Software

Doctorando:

Ángel Leonardo Valdivieso Caraguay (angevald@ucm.es)
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, España

Director:

Luis Javier García Villalba (javiergv@fdi.ucm.es)

Esta tesis doctoral ha sido realizada dentro del grupo de investigación GASS (Grupo de Análisis, Seguridad y Sistemas, grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks) financiado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 (H2020-ICT-2014-2/671672-SELFNET). Asimismo, el presente trabajo ha sido financiado por la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT - Convocatoria Abierta 2012 (Quito, Ecuador). Parte de esta investigación ha sido realizada en el Instituto de Telecomunicaciones de la Universidad de Aveiro.

This thesis is dedicated to my Family:
My Father - Ángel Valdivieso
My Mother - Rosa Caraguay
My Brother - Gustavo Valdivieso

La presente tesis es dedicada a mi familia:
Mi Padre - Ángel Valdivieso
Mi Madre - Rosa Caraguay
Mi Hermano - Gustavo Valdivieso

Acknowledgments

First, I would like to thank my supervisor, Luis Javier García Villalba. Thanks for your support and confidence. Thanks to the support of my family. Ángel, Rosa and Gustavo, your life is a gift of God. I love you.

Thanks to my friends. You guys make my life happier.

Thanks to the members of the GASS research group for all the support I have received during this period.

This work has been done within the Group of Analysis, Security and Systems (GASS, http://gass.ucm.es/), Research Group 910623 from the Universidad Complutense de Madrid (UCM) as part of the activities of the research project funded by the European Commission Horizon 2020 Programme under Grant Agreement number H2020-ICT-2014-2/671672-SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks). This research has also been supported by Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT - Convocatoria Abierta 2012 (Quito, Ecuador).

Agradecimientos

A mis padres Ángel y Rosa y mi hermano Gustavo por todo el amor y apoyo incondicional que me brindan día a día.

A Javier García por la guía, el apoyo y la amistad que me ha brindado todos estos años.

A mis amigos por su sincero soporte y compañía.

Esta tesis doctoral ha sido realizada dentro del grupo de investigación GASS (Grupo de Análisis, Seguridad y Sistemas, grupo 910623 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación SELFNET (Framework for Self-Organized Network Management in Virtualized and Software Defined Networks) financiado por la Comisión Europea dentro del Programa Marco de Investigación e Innovación Horizonte 2020 (H2020-ICT-2014-2/671672-SELFNET). Asimismo, el presente trabajo ha sido financiado por la Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT - Convocatoria Abierta 2012 (Quito, Ecuador).

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3GPP	*Third Generation Partnership Project*
5G	*Fifth Generation*
5GMF	*Fifth Generation Mobile Communications Promotion Forum*
AA	*Active Application*
ABNO	*Application Based Network Operation*
AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
ARPU	*Average Revenue Per User*
BGP	*Border Gateway Protocol*
BSS	*Business Support System*
BWAC	*Broadband Wireless Access and Applications Center*
BWRC	*Berkeley Wireless Research Center*
CLI	*Command Line Interface*
D2D	*Device to Device Communication*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
DPI	*Data Packet Inspection*
DWDM	*Dense Wavelength Division Multiplexing*

EE	*Execution Environment*
EMS	*Element Management System*
EPC	*Evolved Packet Core*
ETSI	*European Telecommunications Standards Institute*
FDL	*Floodlight*
ForCES	*Forwarding and Control Element Separation*
FRP	*Functional Reactive Programming*
HoN	*Health of Network*
IaaS	*Infrastructure as a Service*
ICN	*Information Centric Networking*
IDS	*Intrusion Detection System*
IETF	*Internet Engineering Task Force*
IMS	*IP Multimedia Subsystem*
IoT	*Internet of Thing*
IPR	*Intellectual Property Right*
KPI	*Key Performance Indicator*
LTE	*Long Term Evolution*
M2M	*Machine to Machine Communication*
MCN	*Mobile Cloud Network*
MDSE	*Model driven Software Engineering*
MEC	*Mobile Edge Computing*

MFC	*MobileFlow Controller*
MFFE	*MobileFlow Forwarding Engine*
MIMO	*Multiple Input Multiple Output*
MME	*Mobile Management Entity*
MOS	*Mean Opinion Score*
NaaS	*Network as a Service*
NAT	*Network Address Translation*
NETCONF	*Network Configuration Protocol*
NF	*Network Function*
NFaaS	*Network Functions-as-a-Service*
NFV	*Network Function Virtualization*
NFVI	*Network Function Virtualization Infrastructure*
NFVIaaS	*NFV Infrastructure as a Service*
NFVO	*Network Function Virtualization Orchestrator*
NIC	*Network Interface Card*
NMS	*Network Management System*
NodeOS	*Node Operating System*
NOS	*Network Operating System*
ODL	*OpenDaylight*
ONF	*Open Networking Foundation*
OS	*Operating System*
OSPF	*Open Shortest Path First*
OSS	*Operational Support System*
OTT	*Over the Top*
PaaS	*Platform as a Service*

PAF	*Path Assignment Function*
PGW	*Packet Data Network Gateway*
QFF	*QoE Fairness Framework*
QMOF	*QoS Matching and Optimization Function*
QoE	*Quality of Experience*
QoS	*Quality of Service*
RAN	*Radio Access Network*
RAT	*Radio Access Technology*
RCP	*Routing Control Platform*
RFC	*Request for Comment*
SA	*Situational Awareness*
SBC	*Session Border Controller*
SDMN	*Software-Defined Mobile Network*
SDN	*Software Defined Networking*
SDO	*Standards Developing Organization*
SDR	*Software Defined Radio*
SELFNET	*Self-Organized Network Management in Virtualized and Software Defined Networks*
SF	*Service Function*
SFC	*Service Function Chaining*
SGW	*Serving Gateway*
SLA	*Service Level Agreement*
SME	*Small Medium Enterprise*
SNMP	*Simple Network Management Protocol*
SON	*Self Organizing Networks*

TCO	*Total Cost of Ownership*
TSDB	*Time Series Database*
TTM	*Time to Market*
VIM	*Virtual Infrastructure Manager*
VLAN	*Virtual Local Area Network*
VM	*Virtual Machine*
VN	*Virtual Network*
VNF	*Virtual Network Function*
VNF-FG	*VNF Forwarding Graph*
VPN	*Virtual Private Network*

Abstract

Currently, the growing cost of operational expenditure (OPEX) and capital expenditure (CAPEX) of mobile technologies is constraining the development of new services and applications. The high complexity of the systems requires continuous upgrading and manual re-configuration of the equipments. Operators have to do their best to find and mitigate a big range of problems, such as link failures, security attacks, Quality of Service degradation, software bugs, among others. In some cases, the only solution is the installation of new hardware equipment and the corresponding partial interruption of service and violations in Service Level Agreements (SLA). In this context, the 5G mobile network architecture is expected to support a big range of advanced requirements, in terms of latency, coverage, bandwidth, security and others.

The vision of 5G also includes an integrated self-organized ecosystem of networking, computing and storage resources capable of executing proactive and reactive management tasks. To fulfill this vision, the Self-Organized Network Management in Virtualized and Software Defined Networks SELFNET H2020 project aims to design and implement an autonomic network management framework. The framework will provide self-organizing management capabilities by automatically detect and mitigate network problems that are currently manually solved by network operators. Similarly, SELFNET integrates the self-management paradigm with the use of data mining, learning algorithms, pattern recognition to identify the network behaviour and take actions in order to prevent and minimize several network problems.

For this purpose, this thesis describes aspects regarding Software Defined Networking (SDN), Network Function Virtualization (NFV), 5G, Quality of Service (QoS) and Network Management as connected subjects. It is the belief of this research that SDN and NFV are the key technologies for the development of a new generation of network services and applications. The automatic deployment of SDN/NFV applications provides an autonomic network maintenance delivered by defining high-level tactical measures and enabling autonomic corrective and preventive actions to mitigate existing or potential network problems.

This thesis proposes the SELFNET network monitoring and discovery framework. The framework will enable the automatic deployment of NFV applications in the network infrastructure to facilitate a system-wide, distributed monitoring of the network infrastructure. The research focuses on gathering and storing the information related to physical and virtual devices, cloud environments, flow metrics and SDN/NFV sensors. Similarly, the proposed Framework provides the monitoring data as a generic information source in order to allow the correlation and aggregation tasks.

The present research also includes framework proposals for monitoring and QoS on SDN networks. The monitoring framework uses profiling to provide different monitoring levels based on the requirements of users. Moreover, the pluggable architecture enables the customization of high level metrics. Furthermore, the QoS framework provides functional boxes and interfaces to provide QoS optimization capabilities. The results of experiments demonstrate the quality optimization in comparison with the best effort engine.

Keywords: Monitoring, Network Function Virtualization, SELFNET framework, Software Defined Networking, Self-Organizing Networks.

Resumen

Actualmente, el desarrollo de una nueva generación de servicios y aplicaciones se encuentra limitado por el creciente costo de las inversiones de capital (CAPEX) y gastos operativos (OPEX) en las redes de telecomunicaciones. El alto nivel de complejidad en los sistemas e infraestructura de telecomunicaciones demandan una continua revisión y actualización. De igual manera, los procesos de mantenimiento se realizan únicamente mediante la reconfiguración manual de equipos. Los operadores invierten ingentes recursos en mitigar un gran número de problemas en la red, tales como fallos en los enlaces, ataques de seguridad, degradación de la calidad de servicio, parches de software, entre otros. En algunos casos, la única solución disponible es el reemplazo físico de los equipos ocasionando interrupción parcial o total del tráfico y violaciones en el nivel de servicio contratado. En este contexto, se espera que la nueva generación de infraestructura móvil 5G tenga la capacidad de soportar un amplio espectro de requerimientos en términos de latencia, cobertura, ancho de banda y seguridad.

La nueva visión de 5G incluye un sitema integrado de gestion autónoma de recursos de red, cómputo y almacenamiento. Este sistema debe tener la capacidad de ejecutar tareas de operación tanto reactivas como proactivas. Con este objetivo, el proyecto europeo H2020: Self-Organized Network Management in Virtualized and Software Defined Networks impulsa el diseño e implementación de una arquitectura de gestión autónomo para redes virtuales y definidas por software. La arquitectura agrega capacidades de gestión autónomas que permitan automáticamente detectar y mitigar la mayor cantidad de problemas típicos de red, problemas que actualmente se resuelven únicamente por configuración manual e individual de equipos. De igual manera, SELFNET integra el concepto de gestión autónoma por medio del uso de minería de datos, algoritmos de aprendizaje, reconocimiento de patrones para el análisis del comportamiento de la red que permita identificar problemas de red y ejecutar acciones proactivas y reactivas.

Con este propósito, el presente trabajo estudia la interrelación existente en conceptos tales como: Redes Definidas por Software (SDN), Virtualización de las Funciones de Red (NFV), 5ta Generación de Arquitecturas Móviles (5G), Calidad de Servicio (QoS) y Gestion de Red. El presente trabajo de investigación considera a las Redes Definidas por Software y las Funciones de Red Virtualizadas como tecnologías clave para el desarrollo de una nueva generación de servicios y aplicaciones. La nueva visión de desacoplar el plano de datos y plano de control, de tal manera que se puedan ejecutar aplicaciones de software personalizadas para un determinado fin, abre el camino para el desarrollo de sistemas de gestión autónomos. Asi mismo, se abre la posibilidad de definir un plan de acción de alto nivel para la ejecución de tareas correctivas y preventivas que puedan mitigar problemas de red existentes o potenciales.

La presente tesis doctoral también propone una arquitectura de descubrimiento y monitorización basada en SELFNET. La arquitectura propuesta permite el despliegue distribuido de diversas funciones virtualizadas encargadas de monitorizar el estado de la red. La investigación incluye un modelo de recolección y almacenamiento de métricas relacionadas con infraestructura física, elementos virtualizados, entornos en la nube, flujos

de información y sensores SDN/NFV. De igual manera, la presente investigación tiene como objetivo facilitar las tareas agregación y correlación. Con este fin, la arquitectura transforma la información recibida de fuentes de información heterogenea en un modelo genérico de datos apto para cualquier tipo de sensor o métrica.

Los resultados del presente trabajo también incluye arquitecturas de monitorización y calidad de servicio para redes SDN. La arquitectura de monitorización usa perfiles para el desarrollo y ejecución de múltiples algoritmos basados en el requerimiento de los usuarios. De igual manera, su arquitectura basada en plug-ins permite la inclusión de métricas personalizadas. Por su parte, la arquitectura de mejora en la calidad de servicio propone modulos funcionales e interfaces que permite la ejecución de algoritmos de optimización de QoS. La implementación y resultado de los experimentos verifican que las propuestas presentadas permiten recoletar métricas del plano de datos y también mejorar la calidad de servicio en comparación con el enfoque tradicional del mejor esfuerzo.

Palabras clave: Calidad de Servicio, Gestión de Redes, Monitorización, Redes Definidas por Software, Redes móviles 5G, Virtualización de Funciones de Red.

Part I

State of the Art

Chapter 1

Introduction

Traditional network architectures have experienced several problems with the provisioning of novel network services in terms of operation, security, quality of service, time to market. The close union between data and control planes in network devices, the growing number of network protocols and the private and limited access to the software running in network devices are the main constrains to accelerate the innovation and customization of services. The mitigation of network problems on existing solutions typically require manual re-configuration of the equipment affecting the normal operation of the network. In some cases, the only solution includes the installation of new equipment and functionalities such as routers, *Network Address Translation* (NAT), firewalls, *Data Packet Inspections* (DPIs), among others.

The customization of services requires an open environment where developers can have freedom to modify the network behavior without affecting other users. The solution proposed to solve these challenges is to follow the advances reached by computing, where developers can create their own applications using a high level programming language. The programs can be executed in several equipments thanks to the abstractions of resources provided by the Operating Systems. In this context, the *Software Defined Networking* (SDN) and *Network Function Virtualization* (NFV) appears as a promising strategy to reach these objectives.

SDN proposes the decoupling of data and control planes in network devices enabling their independent development and evolution and a centralized view of the network. For its part, NFV promotes the migration from typical network equipments (DPI, firewall, load balancers) to a software packages or *Network Function* (NF) that can be instantiated in a virtualized infrastructure. Both architectures are complementary and potentially could be integrated to provide an open network environment for developers. However, both concepts are at an early development stage and several challenges are identified before reaching a fully operational technology maturity.

Similarly, in the perspective of the future of *Fifth Generation* (5G) networks and services, the radical decrease of network management expenditures through automation is one of the main relevant challenges. The growing number of mobile devices connected to Internet such as smartphone or tables cannot be further managed with the traditional manual re-configuration of the equipment, for instance to mitigate typical network

problems (link failures, security attacks, *Quality of Service* (QoS) degradation, among others). In this context, the integration of SDN,NFV , Artificial Intelligence, *Quality of Experience* (QoE) to provide a scalable, extensible and smart network management is one of the main objectives of the present research.

Additionally, the present research takes special attention in the automated network monitoring. It is a well-known fact that the decisions taken by management agents depends on the accuracy and quality of the information provided by monitoring agents. The need of a system-wide distributed monitoring architecture for a heterogeneous network infrastructure enable service developers to have more direct and more precise knowledge about the status of the network.

The following sections summarize the main research problems, what were the motivations and the objectives of this study. Then, there is a short summary of the main contributions of this work and how this document is structured.

1.1 Research Problem

In order to provide the identification of the problem, the present section servers as parameters for the discussions found in this work, which will be further detailed in the appropriated sections of this manuscript.

During the research investigation of the literature, it was realized that the development of innovative value-added services is limited by the rigidity of the traditional network architectures. Furthermore, the initial network architecture based on fixed number of computers, each with an IP address, connected to a wired network interface card has evolved towards millions of mobile devices connected to different service provides and requesting low latency and real time services. It represents a paradigm shift in the way that the traditional network management, where operators manually to their best to detect and mitigate network issues, should be replace to a smart autonomic management network services. For this purpose, the 4G standard and the evolution basis towards 5G makes it clear that there is an urgent need of adopting self-management solutions to reduce Opex and Capex.

In this context, this research work is part of the Self-Organized Network Management in Virtualized and Software Defined Networks *Self-Organized Network Management in Virtualized and Software Defined Networks* (SELFNET) H2020 project. SELFNET aims to design and implement an autonomic network management framework. SELFNET will explore SDN and NFV architectures as baseline to provide self-management capabilities for multi-domain heterogeneous networks in order to resolve or mitigate network problems. In this context, the subject of the present research includes the analysis of 5G, SDN, NFV and cloud computing technologies, together with novel algorithms, to achieve a highly intelligent paradigm for smart self-management of complex networking scenarios.

Within the key management tasks provided by SELFNET, the design of an automated network monitoring architecture capable of facilitating the monitoring not only on the basis of the traditional traditional low-level QoS metrics but also in terms of a customizable and extensible set of high-level *Health of Network* (HoN) metrics is challenging. In

this context, the subject of the present research include the automatic deployment of SDN/NFV applications in the network infrastructure to facilitate system-wide, distributed monitoring of the network infrastructure.

This global purpose is covered in three work fronts. Firstly, the monitoring process in SDN environments is analyzed. The optimal behaviour of the SDN network is highly dependent of the accuracy of the information available to take decisions. In traditional architectures, the infrastructure requires the installation of additional equipments and private solutions capable to monitor, collect and process the information provided by each network device. In SDN, an optimal solution that enables SDN controller to obtain information and metrics about the network behaviour in an efficient way is a remarkable contribution. Considering this opportunity, this work started with the idea to use the proper control plane - data plane interface to estimate the network behaviour in real time. This approach offers a cost-efficient solution avoiding the need of additional equipment to deliver these tasks. However, the additional time, load and resource consuming could affect the proper controller behaviour. In this context, it is necessary a model capable to balance the load of monitoring tasks in controller without affecting the accuracy of the metrics.

Secondly, an additional research problem is the provisioning of quality of service in multimedia network applications. In view of the importance of multimedia traffic moving through the networks, the use of SDN approach to provide QoS/QoE is a major opportunity. In this context, it is necessary that a model uses the advantages of a centralized control of the network and the global vision of the traffic to monitor and take decisions to improve the quality of service of a specific type of traffic. Furthermore, the advantages of network virtualization can also be taken into account.

Finally, in the light of the experience gained, the SELFNET Monitoring and Discovery framework is proposed. It includes the SDN and NFV approaches for providing the collection and gathering of several data sources in the context of 5G environments. Specifically, the information collected has a dynamic behaviour and should be available to be processed for multiple purposes (security, QoS, self-healing). In addition to collecting traditional performance metrics, the framework coordinates the creation, instantiation, reconfiguration and removal of network functions. These functions are dynamically allocated in different sections of the virtualized infrastructure.

1.2 Motivation

In general, the emergence of SDN and NFV principles have drawn the attention of the research community. The possibility to overcome the rigidity and complexity of traditional architectures have several opportunities to developers and service providers. SDN and NFV have changed the vision of a traditional monolithic network infrastructure with high maintenance cost and complex upgrade procedures to a more flexible, integrated and interconnected platform of services. These new paradigms can bring innovative value-added services and open new business models with the expected substantial revenues.

Similarly, it is anticipated that the future 5G ecosystem will follow and support the

SDN and NFV as the main enablers to support a flexible environment. In this way, different types of virtualized functions can be deployed, each having a specific purpose, for a particular tenant and sharing the same physical infrastructure. In this context, it is also a strong motivation the inclusion of self-management capabilities for virtualized network environments. In this approach, the NFV functions can be used to automatically resolve or mitigate network problems and improve the QoS/QoE. However, the translation between idea and reality bring additional challenges.

Besides that, it is essential to point out the main level on this purpose: the monitoring and discovery. This research tries to present an architecture that enables the monitoring and discovery for SDN/NFV based environments. It is clear that the success (or failure) of the self-management reactive or proactive operations depends on the effectiveness and accuracy procedures to have knowledge about the status of the network. Similarly, the inclusion of high level Health of Network HoN metrics enable network operators to reduce the amount of collected information.

In this context, the present work aims to provide an automated network monitoring framework capable to collect information of different SDN/NFV based data sources. For this purpose, one particular motivation is the efficient monitoring of SDN networks and the balance between data and control load. Considering this situation, it is also a strong motivation the use of SDN to optimize the QoS in network services. Particularly, the use of real time monitoring data to optimize the routing engine is subject of interest.

1.3 Objectives

This research has one main objective considering the state of the art in 5G, SDN and NFV. This main objective is to use the SDN and NFV principles to provide self-management capabilities and reduce Capex/Opex. Considering the challenges of this main objective, it was divided in three main subjects. The first subject is to optimize the monitoring process in SDN networks. The second objective is to improve the QoS in SDN networks. The third subject is to provide a monitoring and discovery framework for SDN/NFV networks.

In order to fulfill the main objective and its subjects, the following specific activities shall be conducted during this research:

1. Review the state of the art of 5G, SDN and NFV principles.

2. Create and describe the Self-management framework for SDN/NFV networks (SELFNET).

3. Create, implement and test a framework that optimizes the monitoring process on SDN networks.

4. Create, implement and test a framework that improves the QoS in SDN networks.

5. Create and describe the Monitoring and Discovery framework for SELFNET architecture.

1.4 Summary of the Contributions of this Thesis

The results of this thesis is arranged in different fields of knowledge. The main fields of knowledge includes: SDN, NFV, 5G, Network Management and QoS/QoE. The Figure 1.1 summarizes the contributions and the corresponding fields of knowledge.

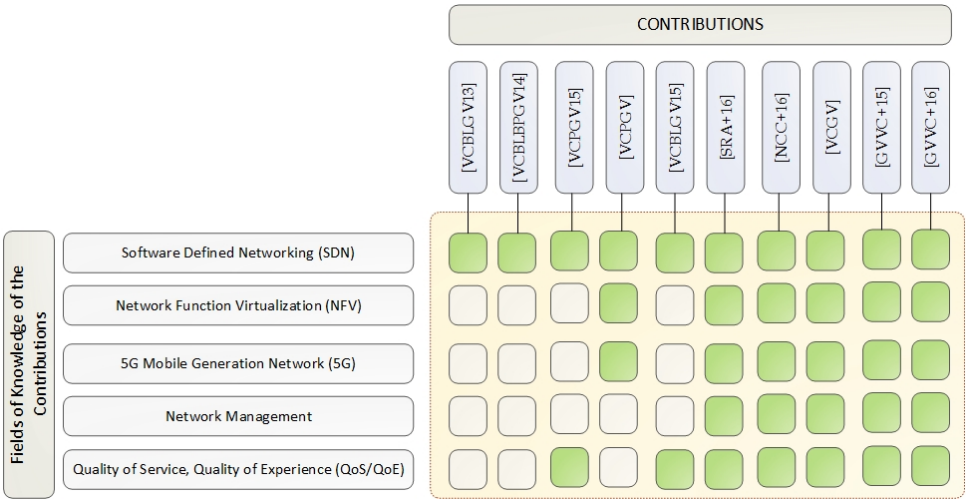


Figure 1.1: Schema of the contributions of this thesis.

In this regard, the contributions focused on SDN were presented in [VCBLGV13] and [VCBLBPGV14]. Then, the contributions published on [VCPGV15] and [VCBLGV15] also involve the topics of QoS/QoE. The contribution presented in [VCPGV] is the first approach that integrates the topics of SDN, NFV for 5G. The rest of the contributions published on [SRA+16], [NVCGV17], [VCGV], [GVVC15], [GVVC16] also includes the network management. In other words, these contributions cover all the topics analyzed on this work.

1.5 Outline of the Thesis

The thesis is organized in two main Sections: State of the Art and Description of the Research.

The State of the Art reviews the recent advances on SDN, NFV, 5G and Related Works. Similarly, the concepts and general architecture regarding SELFNET Project are described. This Section is organized as follows:

Chapter 1 presents a brief explanation of this work consisting of a summary, the research problem and the motivation. Then, the objectives and the summary of the contributions are provided.

Chapter 2 reviews the state of the art regarding Software Defined Networking. This chapter presents a review of the traditional network architectures and the evolution towards SDN. Then, the SDN principles and OpenFlow protocol are described. It also includes a description of the main SDN controller and the ongoing research areas and applications.

Chapter 3 introduces the state of the art of Network Function Virtualization. This chapter reviews the virtualization advances on traditional architectures. Then, an introduction to NFV and the *European Telecommunications Standards Institute* (ETSI)-NFV architecture is described. It also includes a description of the main NFV implementation tools and the ongoing NFV application and use cases.

Chapter 4 describes the state of the art regarding 5G Generation Mobile Network. This chapter presents a review of the 5G global vision and the 5G requirements. Then, the main 5G *Key Performance Indicator* (KPI) are described. It also reviews the 5G future trend and challenges.

Chapter 5 presents the related works. This chapter reviews the advances on monitoring architectures for SDN networks. Then, the research projects on 5G are also described.

Chapter 6 introduces the Self-organized Network Management for SDN/NFV SELFNET framework. This chapter describes SELFNET as a solution for providing advanced network intelligence to SDN/NFV networks. Besides it analyzes the corresponding layers and sublayers.

The Description of the Research corresponds to the original and exclusive contribution of the Doctoral Thesis. It presents the optimized monitoring and QoS routing frameworks. Furthermore, the description of the implementation and the results of experiments are included. Similarly, the design and implementation of the SELFNET monitoring and discovery framework are described. Finally, the conclusions and future work are analyzed. This Section is organized as follows:

Chapter 7 presents two SDN-based frameworks: Optimized Monitoring and Optimized QoS Routing frameworks. This chapter presents the frameworks, the corresponding modules and the implementation.

Chapter 8 summarizes the test results of the frameworks proposed in the previous chapter. In this way, the application scenario and the test results are analyzed.

Chapter 9 proposes the monitoring and discovery framework for SDN/NFV Self-organized networks. This chapter presents the high level architecture and the operational context. Then, the different virtual sensor descriptors and the data sources instances are analyzed.

Chapter 10 summarizes the main conclusions derived from this work. Similarly, the chapter exposes the topics for future research.

1.6 Audience of this Thesis

The prerequisites to access this thesis material are not high. In order to make the work self-contained, we repeat several of the definitions and concepts available in the literature. The reader may need a basic level of knowledge about network protocols, network management and mobile infrastructure. The reviewed bibliography provides the reader with further information where more details can be found regarding the study conducted by this thesis.

Chapter 2

Software Defined Networking

This chapter reviews the main concepts related to SDN. Bearing in mind a broad view, the main applications and challenges are also presented and discussed. This chapter is organized in 6 sections. Section 2.1 reviews the traditional network architectures. Section 2.2 gives details of the separation of data and control planes proposed in SDN. Section 2.3 discusses the OpenFlow architecture. SDN Controllers and Network Operating Systems is the subject of Section 2.4. Section 2.5 presents a review of the SDN applications. Lastly, Section 2.6 summarizes this chapter.

2.1 Traditional Network Architectures

The idea of transmitting information between two points through a network led to the design of communication protocols (TCP/IP, HTTPS, and *Domain Name System* (DNS)) and the creation of specialized devices in the transmission of information. These devices have evolved resulting in a variety of equipment (hub, switch, router, firewall, *Intrusion Detection System* (IDS), middlebox, and filters). This development has produced an exponential increase in the number of connected devices, transmission rate and the emergence of online services (e-banking, e-commerce, e-mail, VoIP, etc.).

All devices responsible for transmitting information have similar features in their design and manufacture. First, there is a specialized hardware in the packet processing (data plane), and over the hardware works an operating system (usually Linux) that receives information from the hardware and runs a software application (control plane). The software contains thousands of lines of code for determining the next hop that a packet should be taken in order to reach its destination. The program follows the rules defined by a specific protocol (there are currently about 7000 *Request for Comments* (RFCs)) or some proprietary vendor technology. Modern equipment also analyses information packets to search malicious information or intrusions (firewalls and IDS). However, all technology or software used in the manufacturing of these devices is rigid or closed to the network administrator.

The administrator is limited only to configure some parameters, usually through low level commands using a *Command Line Interface* (CLI). Moreover, each node is an autonomous system which finds the next hop to be taken by a packet to reach its

destination. Some protocols (*Open Shortest Path First* (OSPF), *Border Gateway Protocol* (BGP)) allow the nodes to share control information between them, but only with its immediate neighbours and in a limited way in order to avoid traditional load on network. This means that there is not a global view of the network as a whole. If the users need to control and modify a particular path, the administrator has to test with parameters, priorities, or uses gadgets to achieve the expected behaviour in the network. Each change in the network policy requires individual configuration directly or remotely from each of the devices. This rigidity makes the implementation of high-level network policies difficult.

Moreover, the policies are required to be adaptive and dynamically react according to the network conditions. As *Operating Systems* (OSs) evolve and adapt to the user needs and technological trends (support multi-CPU, multi-GPU, 3D, touch screen support, etc.), the network adaptability to new requirements (*Virtual Local Area Network* (VLAN), IPv6, QoS, and VoIP) is implemented through protocols or RFCs. However, in the operating system the separation between hardware and software allows the continuous update of application, or even the reinstallation of a new version of an OS. In the area of networks, the design and implementation period of a new idea could take several years until it is published in a protocol and incorporated in new devices. Some services are proprietary of the vendors and require that all network infrastructures belong to the same vendor to work properly. This limitation brings on the dependence on a specific technology or vendor.

2.2 Software Defined Networking

The concept of Software Defined Networking is not new and completely revolutionary; rather it arises as the result of contributions, ideas, and developments in research networking. In [Cal99], three important states are determined in the evolution of SDN: Active Networks (mid-90s to early 2000), separation of data and control planes (2001-2007), and the OpenFlow *Application Programming Interface* (API) and *Network Operating System* (NOS) (2007-2010). All these aspects are discussed below.

2.2.1 Active Networks

The difficulty for researchers to test new ideas in a real infrastructure and the time, effort and resources needed to standardize these ideas on the *Internet Engineering Task Force* (IETF) necessarily give some programmability to network devices. Active networks offer a programmable network interface or API that opens the individual resources of each node for the users, such as processing, memory resources, and packet processing and includes personalized features for the packets that circulate through the node. The need to use different programming models in the nodes was the first step for research in network virtualization, as well as the development of frameworks or platforms for the development of application on the node.

The Architectural Framework for Active Networks v1.0 [Cal99] contains a shared Node Operating System *Node Operating System* (NodeOS), a set of *Execution Environments* (EEs)), and (*Active Applications* (AAs)). The NodeOS manages the shared resources,

while the EE defines a virtual machine for the packet operations. The AA operates within an EE and provides the end-to-end service. The separation of packets to each EE depends on a pattern in the header of incoming packets to the node. This model was used in the PlanetLab [Pla17] platform, where researchers conducted experiments in virtual execution environments and packets were demultiplexed to each virtual environment based on its header. These developments were important, especially in the investigation of architectures, platforms, and programming models in networks. However, their applicability in industry was limited and mainly criticized for its limitations in performance and safety. The work presented in [WT01] is an effort to provide the best performance to the active networks, and the Secure Active Network Environment Architecture [AAKS98] tried to improve their security.

2.2.2 Separation of Data and Control Planes

The exponential growth in the volume of traffic over the network produces the necessity to improve the supervision process and uses best management functions such as the management of paths or links circulating the network (traffic engineering), prediction traffic, reaction, and fast recovery if there are network problems, among others. However, the development of these technologies has been strongly limited by the close connection between the hardware and software of networking devices. Besides, the continuous increase in link rates (backbones) means that the whole transmission mechanism of packets (packet forwarding) is focused on the hardware, separating control, or network management to an application of software. These applications work best on a server, because it has higher processing and memory resources compared with a single network device. In this sense, the project *Forwarding and Control Element Separation* (ForCES) [YDAG04] standardized by the IETF (RFC 3746) established an interface between data and control plane in the network nodes. The SoftRouter [LNR+04] used this software interface to install forwarding tables in the data plane of routers. Additionally, the *Routing Control Platform* (RCP) [CCF+05] project proposed logical centralized control of the network, thus facilitating the management, the innovation capacity, and programming of network. RCP had an immediate applicability because it uses an existing control protocol BGP to install entries in the routing tables of the routers. The separation of data plane and the control plane allows the development of "clean-slate" architectures, such as the 4D project [GHM+05] and Ethane [CFP+07]. 4D architecture proposes architecture of four layers based on functionality: data plane, discovery plane, dissemination plane, and decision plane. Moreover, the Ethane project [CFP+07] proposes a centralized control system of links to business networks. However, the need for custom switches based on Linux, OpenWrt or NetFPGA with support for Ethane protocol made the applicability of this project difficult.

2.2.3 SDN and OpenFlow

At the present time, the OpenFlow protocol [MAB+08] is the most widely used in the research community and it has been the basis of different projects. Companies

like Cisco have also submitted a proposal for a new architecture called Cisco Open Network Environment (Cisco ONE) [KK13]. Simplifying the previous analysis, the term Software-Defined Networking SDN proposes some changes to the networks of today. First, the separation or decoupling of the data plane and control plane allows for evolution and development independently. Secondly, it proposes a centralized control plane, thus having a global view of the network. Finally, SDN establishes open interfaces between the control plane and data plane. The differences between these architectures are shown in Figure 2.1.

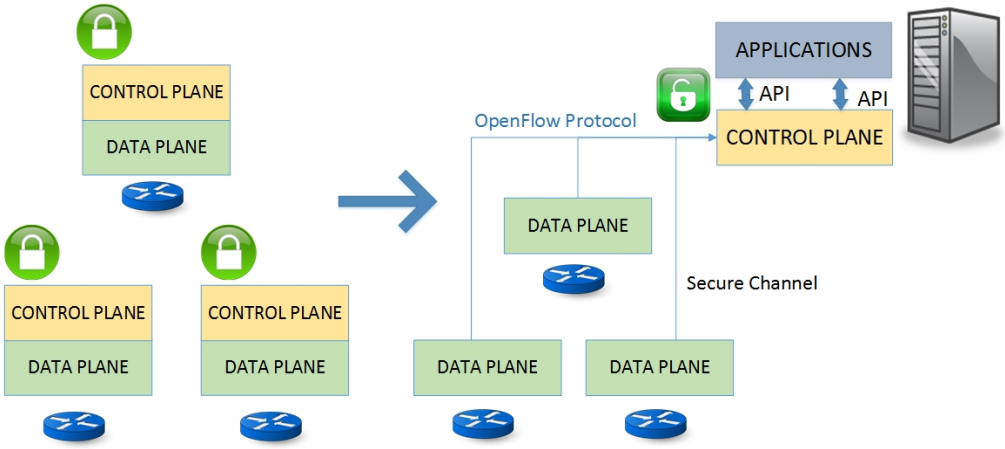


Figure 2.1: Comparison between traditional and SDN architectures

The programmability of the network provided by SDN can be compared with the mobile applications running on an Operating System (Android and Windows Mobile). These applications use the resources of the mobile (GPS, accelerometer, and memory) through the API provided by the OS. Likewise, the network administrator can manage and program resources in the network, according to user needs, through available APIs (proprietary or open) on the controller.

2.3 OpenFlow Protocol

OpenFlow [MAB+08] was originally proposed as an alternative for the development of experimental protocols on university campus, where it is possible to test new algorithms without disrupting or interfering with the normal operation of traffic of other users. Nowadays, the *Open Networking Foundation* (ONF) [ONF17] is the organization responsible for the publication of the OpenFlow protocol and other protocols for SDN, such as OF-Config [ope13].

The advantage of OpenFlow, compared with previous SDN protocols, is the use of elements and features of hardware available in most network devices. These elements are the routing tables and the common functions are as follows: read the header, send the packet to a port, and drop a packet, among others. OpenFlow opens up these elements and functions; so these can be controlled externally. This implies that, with a firmware update, the actual hardware could potentially support OpenFlow. The companies do not need a

complete change of their hardware to implement SDN in their products and services. The OpenFlow architecture proposes the existence of a controller, a switch OpenFlow, and a secure protocol of communication. These elements are shown in Figure 2.2. Each OpenFlow switch consists of flow tables that are managed by the controller. Each flow table has three elements: packet header, actions, and statistics. The packet header is like a mask that selects the packets which will be processed by the switch. The fields used for comparison can be from layer 2, 3, or 4 of the TCP/IP architecture. That means that there is not a separation between layers as in current architectures. All packets processed by the switch are filtered through this method. The number of fields that the switch can process depends on the version of the OpenFlow protocol. In OpenFlow v1.0 [ope09] (the most used version), there are 12 fields, while the latest version OpenFlow v1.3 defines the existence of 40 fields including support for IPv6.

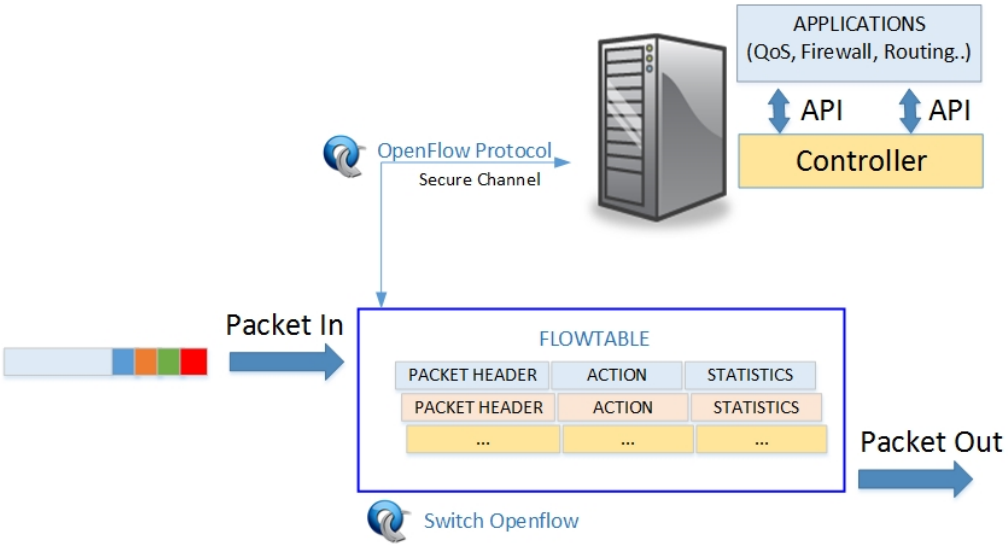


Figure 2.2: OpenFlow architecture

Once the header of an incoming packet matches the packet header of the flow table, the corresponding actions for that mask are performed by the switch. There are main and optional actions. The main actions are as follows: forward the packet to a particular port, encapsulate the packet and send it to the controller, and drop the packet. Some optional actions are as follows: forward a packet through a queue attached to a port (enqueue action) or 802.1D processing capabilities. If the header of an incoming packet does not match with the packet header of the flow table, the switch (according to its configuration) sends the packet to the controller for its analysis and treatment. Finally, the statistics field uses counters to collect statistic information for administration purposes.

An OpenFlow switch manages three kinds of table: flow, group and meter tables. The controller is able to add, delete or update flow entries in a flow table. In turns, each flow entry consists of the following elements: match fields (for matching packets), counters (tracking packets), priority, timeouts and a set of instructions. An OpenFlow switch uses a pipeline in order to define how the packets interact with flow tables, as is depicted in Figure 2.3.

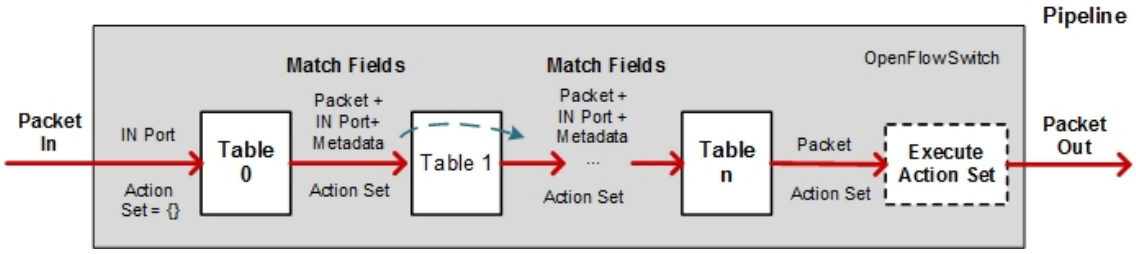


Figure 2.3: OpenFlow pipeline

The OpenFlow protocol defines the following types of messages between the switch and the controller: controller to switch, symmetric, and asynchronous. The messages type controller to switch manage the state of the switch. Symmetric messages are sent by the controller or switch to initiate the connection or interchange of messages. The asynchronous messages update the control of the network events and the changes of state switch. Similarly, OpenFlow establishes two types of switches: OpenFlow-only and OpenFlow-enabled. OpenFlow-only switches use only OpenFlow protocol to process packets. On the other side, OpenFlow-enabled switches can additionally process the packet using traditional algorithms of switching or routing.

The controller receives the information from the various switches and remotely configures the flow tables of the switch. Here, the user can literally program the behaviour of the network. Unlike active networks, which proposed a "Node Operating System" OpenFlow opens the notion of a NOS. In this respect, in [FRZ14], the NOS is defined as the software that abstracts the installation of the state in the switches of network of the logic and applications that control the behaviour of the network. In recent years, the NOS has evolved according to the needs and applications for researchers and network administrators.

2.4 SDN Controllers

The concept of NOS is based on the function of an operating system in computing. That is, the Operating System allows user to create applications using high-level abstraction of information, resources, and hardware. In SDN, some authors [SSHC+13], [RFR+12] have classified the abstractions of network resources as southbound and northbound interfaces (Figure 2.4). The function of the southbound interfaces is to abstract the functionality of the programmable switch and connect it to the controller software. A clear example of southbound interface is OpenFlow. On the southbound interfaces, you run a Network Operating Systems. An example of NOS is NOX [GKP+08a], among others. On the other hand, the northbound interfaces allow applications or high level network policies to be easily created and they transmit these tasks to NOS. Examples of these interfaces are Frenetic [FHF+11], [FGR+13], Procera [KF13], [VKF12], Netcore [MFHW12], and McNettle [VW12].

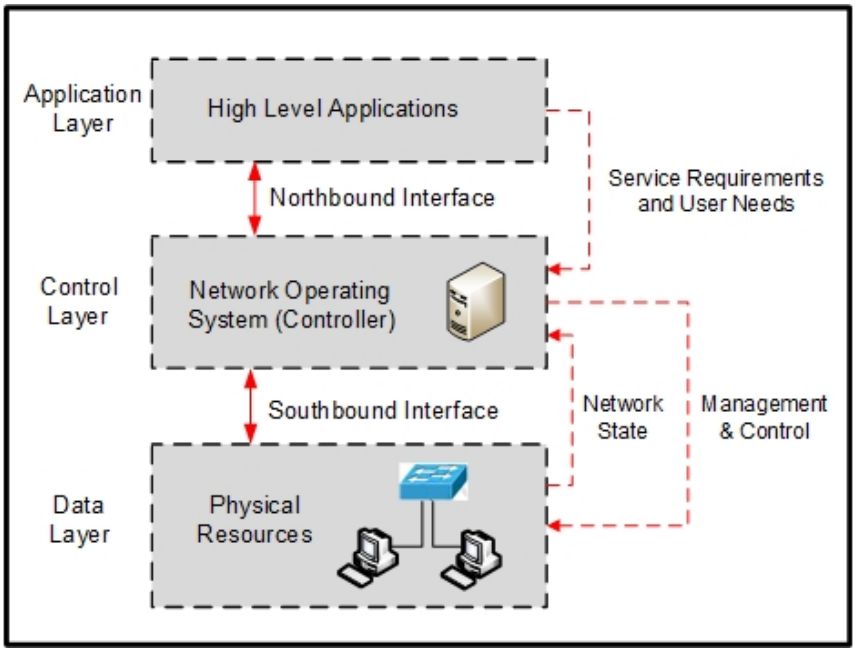


Figure 2.4: NOS southbound and northbound

2.4.1 Network Operating Systems

The NOX software [GKP+08a] is the first NOS for OpenFlow and consists of 2 elements: processes of controller (controller) and a global view of the network (network view). Depending on the current state of the network, the user can make decisions and set the network behaviour through these processes. In NOX, traffic is handled at the level of flows (flow-based granularity); that is, all packets with the same header are treated similarly. The controller inserts, deletes entries, and reads the counters found in the flow tables of the switches. Furthermore, due to the dynamic nature of traffic, NOX uses events (event handlers) that are registered with different priorities to be executed when a specific event occurs in the network. The most used events are switch join, switch leave, packet received, and switches statistics received. Additionally, NOX includes "system libraries" implementations and common network services. Finally, NOX is implemented in C++ providing high performance. Moreover, there is an implementation entirely in Python denominated POX, which provides a more friendly developed language.

Floodlight (FDL) [flo17] is a Java-based OpenFlow controller. The core of FDL is an evolution of a previous OF-controller known as Beacon [Eri13]. FDL provides a simple and unrestricted interface; that is, the user can freely use the constructors available in Java (threads, timers, sockets, etc.). Furthermore, FDL is a NOS based on events; that is, the user sets the events that the controller listens to. The interaction with OpenFlow messages of the switch is done by the library OpenFlowJ, an implementation of the OpenFlow 1.0 [ope09] protocol and a Provider interface that contains the following listeners: IOFSwitchListener, IOFInitializerListener, and IOFMessageListener. Additionally, FDL has multithreading support and provides important APIs implementations (Device Manager, Topology, Routing, and Web UI) as well as the ability to start, add, and complete

applications without completely terminating a process in FDL (runtime modularity).

OpenDaylight (ODL) [MVTG14] is a NOS that uses *Model driven Software Engineering* (MDSE) and model driven network management in order to provide flexibility and scalability in the development of SDN applications. In this way, users can include multiple services and application using the ODL southbound plugins. Modelling language such as YANG together with NETCONF/RESTCONF protocols are used to facilitate developers the design of network services/applications. For its part, ONOS [BGH+14] is a NOS focused on providing a distributed SDN control platform. For this purpose, it proposes a distributed architecture in order to ensure availability and scale-out. Similarly, it provides a global network view and a logically centralized control even though the servers can be distributed across multiple locations.

2.4.2 High Level Network Policies Languages

Although a NOS can handle the flow tables of the switches, there are some problems that can cause malfunction of the network [SSHC+13], [RFR+12], [GRF13]. For example, the controller receives the first packet that arrives at the switch and has not matched a header in the flow table. Then the controller analyses it, assigns actions, and forwards these instructions to the switch so that the other similar packages follow the same route. However, during this time, the second, third, or fourth similar packets can be received by the controller and cause an erratic operation. In other words, there are virtually two processes running, one on the controller and another on the switch, and these processes are not fully synchronized.

Another limitation is the composition; that is, if the user wants to configure two different services on the same switch (e.g., routing and monitoring), it is necessary to manually combine the two actions on the switch, prioritize, and keep the semantics of each element of the network. This makes the design, coordination, and reuse of the libraries very difficult. Additionally, the switch has to handle two types of messages simultaneously: packets and control messages. Any mismatch can cause a packet to be processed with an invalid policy and thereby causing major security problem on the network. For example, if there are two entries in a flow table with the same priority, the switch behaviour might be non-deterministic, because the execution would depend on the design of the switch hardware. For this reason, the research community has worked on secure interfaces that automatically interact and coordinate the correct behaviour of the switch (northbound).

Procera [KF13], [VKF12] is a framework that allows politics or high-level network configurations to be expressed. This architecture provides different actions and control domains to program the behaviour of the network. The main domains of control are as follows: time, data usage, flow, and status. With these domains, the user can determine a behaviour depending, for example, on the time of day, amount of data transmitted, privileges or groups of users, type of transmitted traffic, and so forth. Actions can be temporal or reactive and are expressed on a high-level language based on *Functional Reactive Programming* (FRP) and Haskell. In [VKF12] are the details of this language as well as examples of using Procera in monitoring applications and users control on a college campus. For its part, Frenetic [FHF+11], [FGR+13] is a high-level language dedicated to

SDN networks developed in Python. It is structured by 2 sublanguages: a Network Query Language and a Reactive Network Policy Management Library. The Network Query Language allows the user to read the status of the network.

This task is performed by installing rules (low-levels rules) on the switch which does not affect the normal operation of the network. In addition, the Network Policy Management Library is designed based on a language for robots, Yampa, [CNP03] and web programming libraries in Flapjax [MGB+09]. The actions use a constructor type rule containing a pattern or filters and action list as arguments. The main actions are as follows: sending to a particular port, sending packet to the controller, modification the packet header, and blank action that is interpreted as discard the packet. The installation of these policies is performed by generating policy events (queries), primitive events (Seconds, SwitchJoin SwitchExit, and PortChange), and listener (Print and Register). The results of experiments [FHF+11] shows that Frenetic provides simplicity and a significant savings in code and lower consumption of network resources compared to NOX.

One of the additional advantages of this language is the composition; that is, independent functional modules can be written and the runtime system coordinates its proper function in the controller and the switch. There are 2 types of composition: sequential and parallel. In sequential composition, the output of one module is the input of the next, for example, a load balancer that first modifies the IP destination of a packet and then searches the output port according to the new IP header. In parallel composition, both modules are executed virtually simultaneously in the controller; for example, if the balancer sends a packet with destination IP A to port 1, and packet B IP destined to port 2, this composition would result in a function that sends incoming packets for ports 1 and 2.

McNettle [VW12] is a controller specially designed to offer high scalability at the SDN network. This is achieved using a set of message handlers (one for each switch) having a function that handles the switch-local and network-state variables and manages the supply actions from the network flows. The idea is that the messages from the same switch are handled sequentially, while messages from different switches are handled concurrently. Similarly, each message is processed in a single core CPU to minimize the number of connections and synchronizations inter-cores among other performance improvements. The tests performed in [VW12] show that McNettle have a higher multicore performance compared to NOX or Beacon. The controller proposed in [GRF13] is based on the verification of the established politics, instead of searching bugs monitoring the controller operation. To perform the verification, the first step is to make use of the high-level language Netcore [MFHW12] to describe only the network behaviour. Then, the Netcore Compiler translates the politics to network configurations as flow table entries. The flow tables information is analysed by the Verifier Run-time System which transforms the network configuration into a lower abstraction level named Featherweght OpenFlow. Featherweght Openflow is a model that use synchronization primitives to guarantee the coherent behaviour of the flowtables. Additionally, the Kinetic tool is described in [RFR+12]; this tool allows performing consistent updates in the network using two mechanisms: per-packet consistency and per-flow consistency. The per-packet consistency

mechanism ensures that a packet that is transmitted across the network is processed with the same configuration when an update occurs. The per-flow consistency mechanism ensures that every packet that belongs to the same flow (e.g., a TCP connection) will be processed in the same way by every switch in the network.

2.5 SDN Applications

Software-Defined Networking provides the ability to modify the network behaviour according to user needs. In other words, SDN itself does not solve any particular problem, but provides a more flexible tool to improve the network management. In order to test the advantages of this architecture, the research community has presented multiple projects of interest. Next, some of these applications are described.

2.5.1 Home Networking

In the emerging topic of *Internet of Thing* (IoT), the management of devices and network resources in home networks is a big challenge due to the number of users and devices connected to the same point (usually an access point). In [KF13], [KSC+11], the authors present an implementation of an OpenFlow-based system that allows the monitoring and management of user and control of the Internet access based on "usage caps" or a limited data capacity for each user or device. The system provides visibility of the network resources and management of access based on user, group, device, application, or time of day and even enables the ability to exchange data capacity with another user. The system of control and network monitoring uses the friendly interface Kermit. The capacity management and network policies are based on the Resonance language [NRFC09].

2.5.2 Security

The global vision of the network can improve the security of the systems. This security cannot be based only in the host-security, because such defenses are ineffective when the host is compromised. In [RMTF09], the Pedigree system is presented as an alternative to provide security in the traffic moving in an enterprise network. This OpenFlow-based system allows to the controller the analysis and the approval of connections and traffic flows in the network. The host has a security module in the kernel (tagger) that is not under users control. This module labels the connections request to send information through the network (processes, files, etc.). This label is sent to the controller (arbiter) in the start of the communication. The controller analyses the tagger and accepts or rejects the connection according to its policies. Once the connection is authorized, the corresponding flow tables are installed in the switch. Pedigree increases the tolerance to a variety of attacks, such as polymorphic worms. The systems increase the load in the network traffic and the host. However, this load is not higher than common antivirus software.

2.5.3 Mobile Networks

The devices in the infrastructure on mobile carrier networks share similar limitations as computer networks. Likewise, the carrier networks execute standards and protocols, for example, the *Third Generation Partnership Project* (3GPP) Project as well as the private vendor implementations. At this point, the SDN paradigm and its flow-based model can be applied on this kind of infrastructure offering better tools. *Software-Defined Mobile Network* (SDMN) [PWH13] is an architecture that enables openness, innovation, and programmability to operators, without depending on exclusive vendors or *Over the Top* (OTT) service providers. This model consists of two elements: *MobileFlow Forwarding Engine* (MFFE) and the MobileFlow Controller *MobileFlow Controller* (MFC). MFFE is a simple and stable data plane and with high performance. It has a more complex structure than an OpenFlow switch, because it must support additional carrier functions, such as layer 3 tunnelling (i.e., GTP-U and GRE), access network nodes functions, and flexible charging. The MFC is the high performance control plane, where the mobile networks applications can be developed. Additionally, MFC has 3GPP interfaces to interconnect with different *Mobile Management Entitys* (MMEs), *Serving Gateways* (SGWs), or *Packet Data Network Gateways* (PGWs).

2.5.4 Multimedia

The multiple online multimedia services, for example, the real time transmissions, require high levels of efficiency and availability of the network infrastructure. According to studies presented by CISCO, the IP video traffic will grow from 70% in 2015 to 82% by 2020 [zet16]. Moreover, in the last years, the concept of QoE [LCMP12] gained particular strength, which attempts to redefine the QoS considering the level of user acceptance to a particular service or multimedia application. Therefore, SDN allows the optimization of the multimedia management tasks. For example, in [KSKD+12] is improved the QoE through the path optimization. This architecture consists of two elements: the *QoS Matching and Optimization Function* (QMOF) that reads the different multimedia parameters and establishes the appropriate configuration for this path, and the *Path Assignment Function* (PAF) that regularly updates the network topology. In case of degradation of the quality on the links, the system automatically modifies the path parameters taking in count the priorities of the users. Similarly, the project OpenFlow-assisted *QoE Fairness Framework* (QFF) [GEB+13] analyses the traffic in the network and identifies the multimedia transmissions in order to optimize them in function of the terminal devices and the network requirements.

2.5.5 Reliability and Recovery

One of the most common problems in the traditional networks is the hardness to recover a link failure. The convergence time is affected by the limited information of the node to recalculate the route. In some cases, it is necessary the intervention of the network administrator to reestablish the network datapath. At this point, the global vision of SDN enables the customizing of recovery algorithms. [SSC+12] proposed an OpenFlow-based

system that uses the mechanism of restoration and protection to calculate an alternative path. In restoration mechanism, the controller looks for an alternative path when the fail signal is received. Meanwhile, in protection the system anticipates a failure and previously calculates an alternative path. Similar to a failure on switch or routers, the malfunction of the SDN controller (NOS failure, *Distributed Denial of Service* (DDoS) attack, and application error) can cause a collapse of the whole network. Therefore, the reliability of the network can be ensured through backup controllers. However, it is necessary to coordinate and update the information of control and configuration between principal and backup controllers. The CPRecovery [FBMP12] component is a primary backup mechanism that enables the replication of information between primary and backup controller. The system uses the replication phase to maintain the updated backup controller and the phase of recovery that starts the controller backup at the moment it detects a failure of the principal controller.

2.6 Summary

This chapter summarizes the SDN architecture. First, the limitations of traditional network architectures are analysed. Then, the differences between traditional and SDN architectures are described. Next, OpenFlow as the most relevant SDN southbound protocol is reviewed. Finally, the principal Network Operating Systems and the relevant applications are presented.

Chapter 3

Network Function Virtualization

This chapter reviews the main concepts of NFV. This chapter is organized in 5 Sections. Section 3.1 reviews the virtualization concept in traditional architectures. Section 3.2 gives details on the NFV concept. Section 3.3 discusses the main NFV implementation tools. NFV applications and use cases is the subject of Section 3.4. Lastly, Section 3.5 summarizes this chapter.

3.1 Virtualization in Traditional Architectures

In traditional architectures, the infrastructure is composed by a large number of network devices, each operating their own private software and highly dependent in proprietary hardware. For this reason, the design and installation of new services usually require the individual software updating or the replacement of hardware. This rigidity increases the installation and operational costs.

In order to optimize the reuse of the available resources, the idea of share the infrastructure between different users, each with their private logical isolated space have won the attention of companies and service providers. In this context, similar to computer virtualization layer, where a hardware abstraction permits slicing and sharing the hardware resources with different OS in a host, the goal of network virtualization is to isolate multiple logical networks, each of them with completely different addressing and forwarding mechanism, but sharing the same physical infrastructure. In other words, in network virtualization, it is intended that multiple virtual networks can operate on the same infrastructure, each with its own topology and routing logic.

One of the first approaches on network virtualization is the VLAN [TFF+13] technologies. In VLAN networks, differents users can share network infrastructures. However, the separation is controlled only by the network administration and with limited parameters (port number) and just work with known network protocols. The network administrator assign a VLAN as a logical network formed by a group of hosts in a single broadcast domain. The broadcast domain and the corresponding logical topology is different from the physical network topology. VLANs opened facilities for management and reconfiguration of networks. However, the number of VLANs is limited to 4094 due the rigidity of the protocol.

Another approach in the development of virtualization solutions is the *Virtual Private Network* (VPN) [CB10]. A VPN creates a secure tunnel to communicates multiple sites. The secure tunnel is carried over a public network and the nodes can be geographically separated. The type of VPNs depends on the layer of transportation to be protected. For instance, the layer 1 VPN corresponds to the circuit switching domain or layer 3 VPN over a L3 protocol such as IP. In addition, there are other solutions that provides virtualization capabilities. The Active Networks opens the remote control of the network device and enables some programmability resources in network devices. This remote control of the network functions can be used to filtering the traffic to different users and provide virtualization capabilities. Similarly, the OpenvSwitch [PPA+09] open source tools allows the creation of virtual environments in software switches. This virtual switch can be configured to provide private network domains to different virtual machines in the same hardware. Furthermore, device manufacturers also includes private virtualization solutions in the *Network Interface Card* (NIC).

3.2 Network Function Virtualization

As outlined before, the virtualization is not a new concept. The virtualization refers to the abstraction of the logical resources from the physical resources, creating multiple logical instances over the same physical infrastructure [JP13]. In this context, the virtualization has reached different technical domains: virtualization of operating systems, computer hardware platforms, storage capacities and networks. In the field of networking, the virtualization was originally understood as enabling simultaneously multiple *Virtual Networks* (VNs) over a physical network. However, the virtualization principles also has been extended to other functionalities of networks.

Nowadays, the telecom providers has several challenges and high costs in order to update or install new network functions or appliances. Often, the appliances (e.g. firewall, DPI) are deployed in propietary hardware or private software, and consequently, cannot be reused or modified by other service providers. Moreover, the rigidity and complexity of network deployments reduce the customization capabilities of the services provided by operators.

In this context, the Network Function Virtualization proposes the transferring of the different NF (routing, firewall, DPI, gateway) as virtual software-based applications executed in IT platforms (servers, switches and storage) [MSG+16]. This new vision of IT services provides a major flexibility and scalability, facilitates the development cycles and reduce costs [CDLL15]. Figure 3.1 describes the differences between NFV and traditional architectures.

3.2.1 ETSI-NFV Architecture

The Service Providers and the ETSI, that includes more than 28 network operators and 150 telecommunication enterprises, has released the first NFV specification [ETS13b]. The virtualization of NFs enables the running of virtual services on standard switches, storage

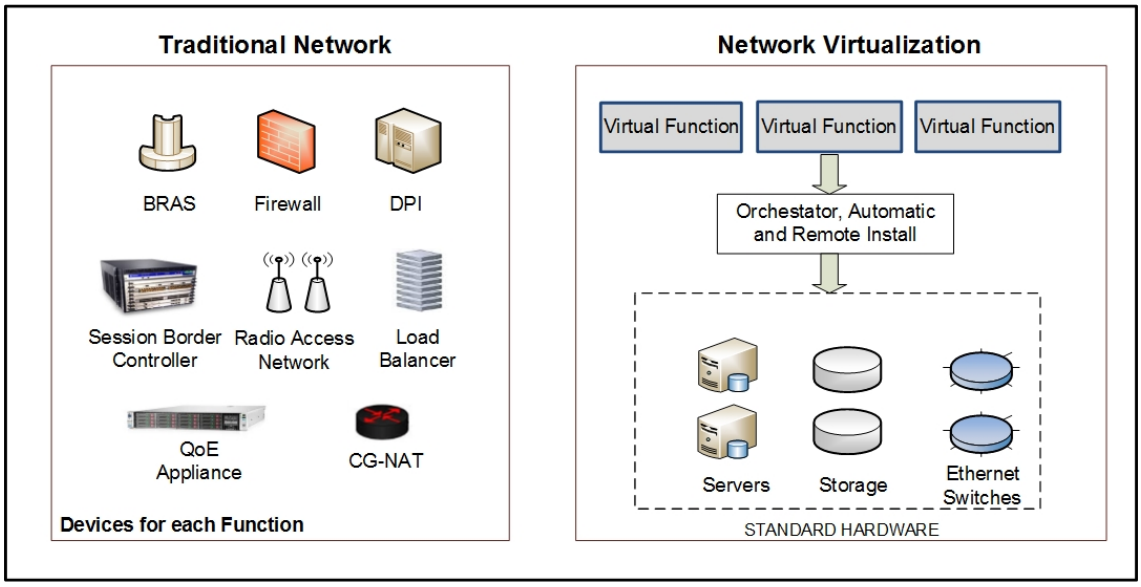


Figure 3.1: Traditional architectures vs. function virtualization

or high-volume servers. Examples of NFs includes switching elements, mobile or sensor network nodes, traffic analysis functions, content distribution functions, *Session Border Controller* (SBC) and others. This new approach accelerates the deployment of new NFs, faster *Time to Market* (TTM), reduce of CapEx and OpEx and improves the customization and elasticity of services.

The NFV architecture released by ETSI is described in Figure 3.2 [ETS13a]. The NFV is composed of three main modules: *Network Function Virtualization Infrastructure* (NFVI), *Virtual Network Function* (VNF) and NFV Management and Orchestration (NFV M&O).

- **NFVI**. It represents the hardware and software resources of the system where the NFV concept are applied. Computing, storage and networking resources are included. The virtualization layers abstract the different resources and enable the isolation and independence of virtual compute, virtual storage and virtual network for different users.

- **VNF**. Represents an instance of a NF that runs over the NFVI. The VNF also includes the corresponding operational and management systems (*Operational Support System* (OSS) *Business Support System* (BSS)).

- **NFV M&O**. The principal function is the orchestration and management of the VNF and NFVI ensuring the optimal and effective operation of the Virtual Functions in the available infrastructure. The principal NFV M&O components are: the orchestrator, a VNF manager and the *Virtual Infrastructure Manager* (VIM). The VIM uses a resource inventory to control the availability of the different resources which guarantees the provisioning of services [ETS14].

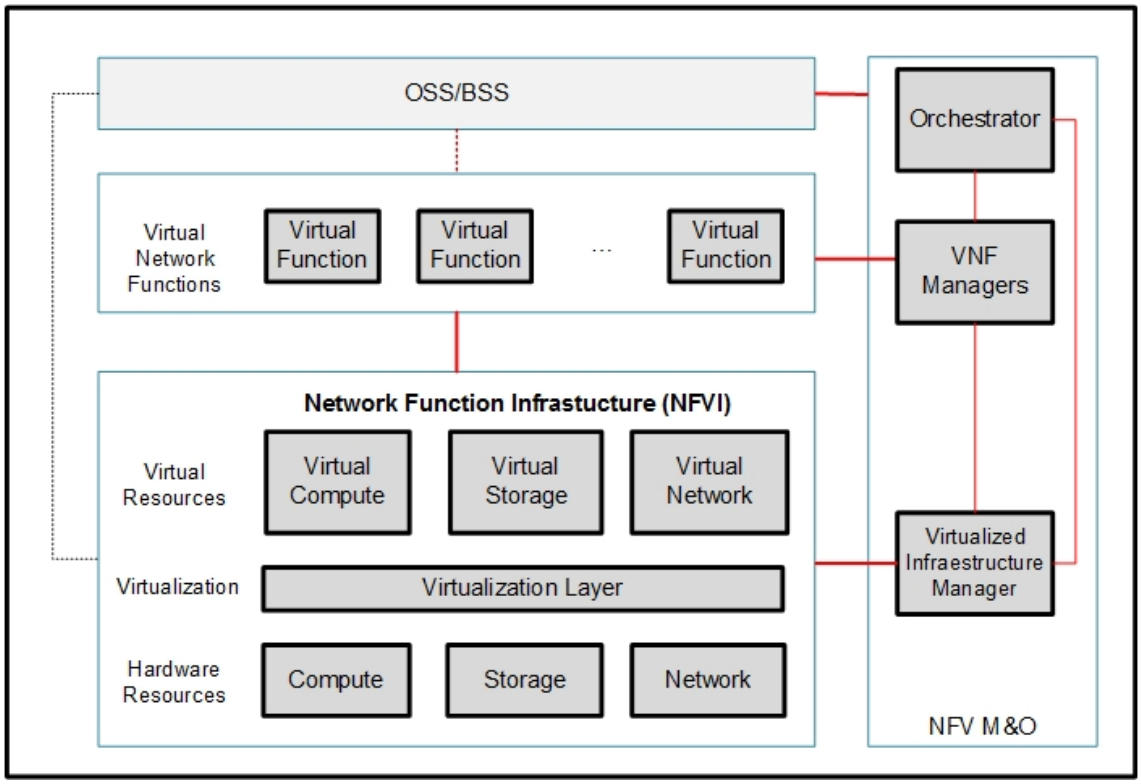


Figure 3.2: NFV architecture

3.2.2 IETF Service Function Chaining SFC

The *Service Function Chaining* (SFC) is defined as the interconnection of service instances, termed service chaining, works by mapping packets to service chains at the edges and forwarding them between service instances [BRL+14]. In other words, SFC enables the organization and ordering of execution of multiples service functions that will be carried out by network devices during the transfer of information.

IETF establishes the IETF SFC Work Group (WG) which concentrates their efforts in the development of an open SFC architecture. IETF defines SFC as ordered set of abstract service functions and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification [HP15]. The IETF SFC architecture is depicted in the Figure 3.3.

The IETF SFC architecture [HP15] is composed of the following modules: *Service Function* (SF), Service Classification Function SCF, SFC-encapsulation and Service Function Forwarder.

- **SF**. It provides a specific processing in the received packets. The treatment can be executed at various ISO layers. The SF can be implemented as a virtual or physical network element. Similarly, a network element can provide multiple service functions and multiple SF occurrences can be executed in the same administrative domain.

- **SFC**. It receives the incoming traffic and splits them depending on a specific classification criteria. Its granularity depends on the SFC capabilities and the

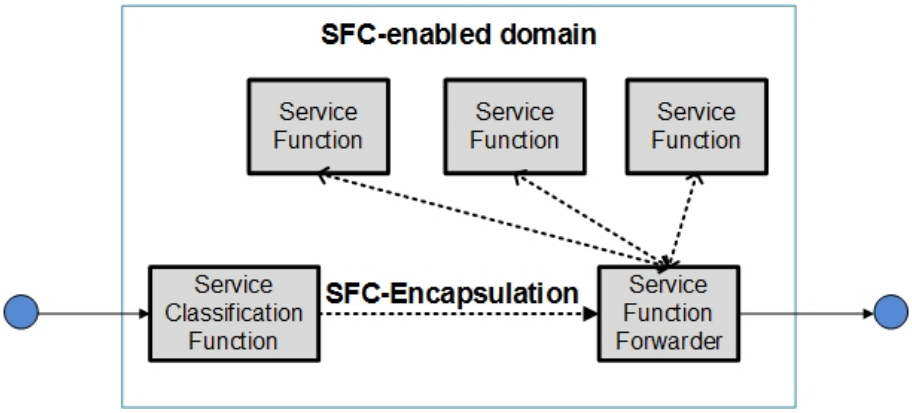


Figure 3.3: IETF SFC architecture [GMUJ16]

rigidity of the classification policies. Each of these traffic domain is known as Service Function Path SFP. Then, each SFP is sent to the SFC-encapsulation.

- **SFC-encapsulation**. SFC-encapsulation: It provides the required information to clearly identify a SFP during the function chaining process. The encapsulation information includes a SFP identification and the related metadata.
- **SFF**. It uses the information provided by the SFC-encapsulation to forward the traffic to the corresponding service functions. Then, once the traffic is processed, the SFF collects and transport the results of the SF to another SFF and finally terminates the SFP.

3.3 NFV Implementation Tools

The NFV principles completes the set of requirements to provide users a complete stack of cloud solutions. For this reason, open source and private cloud solutions have integrated NFV architecture in their products and tools [MVTG14], [KT217]. Although there is not a complete NFV open source tool, in this sections some advances on the implementation of different NFV elements are described.

3.3.1 OpenStack

OpenStack [ope17b] is the most widely open source tool used for cloud computing service development. Originally, OpenStack was developed by NASA and designed to provide an *Infrastructure as a Service* (IaaS) capabilities to service providers. However, its features have been extended to include the NFV capabilities in order to address additional advanced networking challenges.

The OpenStack architecture is depicted in Figure 3.4. The main components of OpenStack are: Compute, Networking and Controller.

- **Compute**. It is responsible of providing the requested compute instances. In this way, the OpenStack compute (also known as Nova) guarantees the spawning, scheduling and decommissioning of the requested virtual machines.

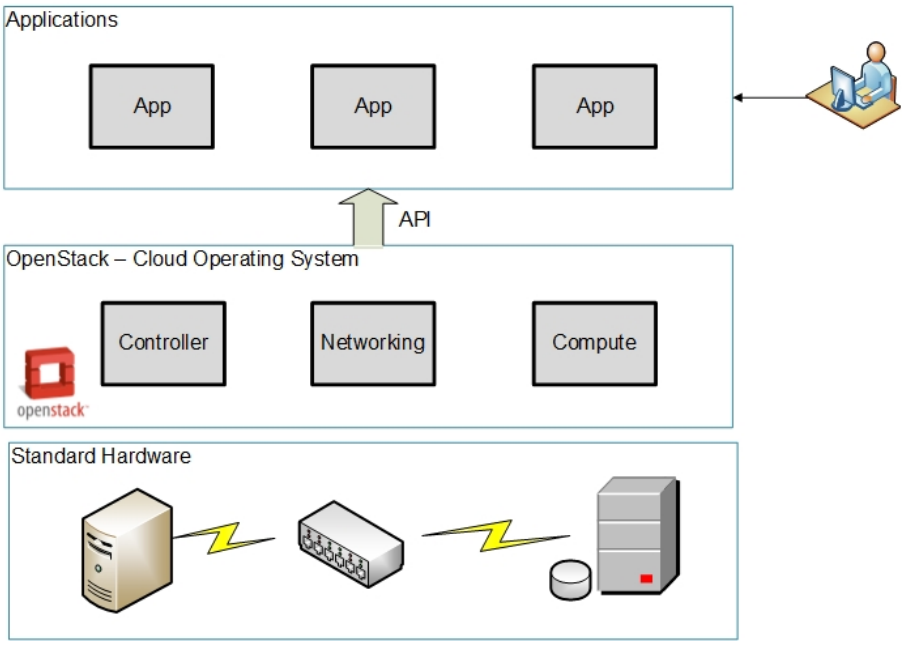


Figure 3.4: OpenStack architecture

- **Networking**. It provides the network connectivity (Network-Connectivity-as-a-Service) for the different OpenStack services. For instance, OpenStack Networking (also known as Neutron) enables the connectivity between the instances of compute VMs. In this way, the Neutron API allows the creation and management of the virtual networks and the architecture support multiple vendors and technologies.
- **Controller**. The controller includes different services that enables the provision of OpenStack services. For instance, Identity service, Image Service, Management elements of Compute and Networking or additional optional services. The number of controller components depends on the particular OpenStack deployment and requirements.

The relationship between OpenStack and NFV is that the services provided by OpenStack can be considered as the management of the virtual infrastructure (Virtual Infrastructure Manager) [JP13].

3.3.2 OpenBaton

OpenBaton [ope17a] is a ETSI NFV compliant *Network Function Virtualization Orchestrator* (NFVO). In other words, it coordinates the lifecycle of NFVs in the available virtual infrastructure. OpenBaton provides a Network Service Management using different Virtual Network Function Managers VNFMs. Similarly, its pluggable architecture enables the supporting of different VIM types, the easy integration with OpenStack and the integration with runtime management of Network Services.

The OpenBaton architecture is depicted in Figure 3.5. The principal components are: *Element Management System* (EMS), Virtual Network Function Manager VNFM, NFVO.

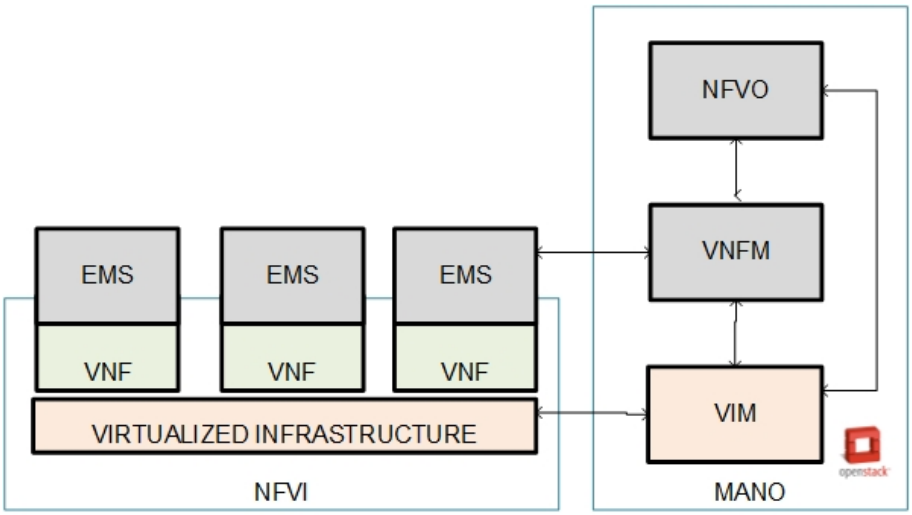


Figure 3.5: OpenBaton architecture

- **EMS**. It is an agent deployed inside the VMs that receives and processes the lifecycle events sent by the VNFM. For this purpose, it implements an *Advanced Message Queuing Protocol* (AMQP) producer/consumer. In this context, it subscribes to RabbitMQ message bus to receive the register-queue attending further commands from the VNFM.

- **VNFM**. It interoperates with the EMS in order to control the lifecycle of a VNF. The assigned management tasks can be realized in a single VNF instance or in a group of multiple and different types of VNFs. In this context, the generic VNFM communication is realized using AMQP over RabbitMQ.

- **NFVO**. It receives the request and controls the on-boarding process of a new network service or VNF instance in the virtualized infrastructure. The orchestration process includes the following tasks: NS lifecycle management, resource management, validation and authorization of NFVI requests, among others.

3.4 NFV Applications and Use Cases

The ETSI Industry Specification Group NFV-ISG [ETS] exposes a variety of NFV use cases. Depending on the application field, the NFV has several terminology definitions. The *NFV Infrastructure as a Service* (NFVIaaS) enables the deployment of NFs in the infrastructure of an external service providers. For its part, the VNF as a service (VNFaaS) allows the use or lease of a VNF to the infrastructure of private service providers. The VN platform as a service (VNPaaS) leases both the infrastructure and the available applications. The customers VNF can also be instantiated. The *VNF Forwarding Graph* (VNF-FG) facilitates the service chaining due to the creation of a logical path with the hops NFs to deploy a service. Next, some of these applications are described.

3.4.1 Mobile Network Virtualization

The growing number of mobile devices and the increasing demand of high data rates and slow latency have pressured service operator to continally upgrade and enhance their infrastructure. In this context, the possibility of virtualizing the different functions and operations of a mobile infrastructure is gaining the attention of the research community [AHGZ16]. In [HSMA14], the authors faces the technical challenges and advances on virtualization of mobile networks. Similarly, they proposes an approach to the separation of data and control planes in mobile entities. In this way, the virtualization of an eNodeB (*Long Term Evolution* (LTE) base station) could reduce the number of operations required to connect the eNodeB to the mobile core network. Moreover, a virtual eNode could offer a centralized computing infrastructure for multiple base stations enabling the sharing of resources between different service providers. For its part, the virtualization of the mobile core network *Evolved Packet Core* (EPC) has the potential to reduce the OPEX and CAPEX. An NFV-based EPC infrastructure could be scaled on-demand in real time and can be easily updated to support a variety of access technologies.

3.4.2 Optical Networks

The management of optical transport networks has evolved from a *Dense Wavelength Division Multiplexing* (DWDM) to dynamic switching technologies and flexible grid transmission schemes. In order to achieve this objective, the traditional *Network Management System* (NMS) for transport networks is improved by the flexibility provided by NFV approach. In [KFG15], the authors proposes a novel framework for the customized control of resources with a given user bandwidth demand. It uses flexi-grid, SDN/NFV and *Application Based Network Operation* (ABNO) to provide resilient and elastic network capabilities depending of actual and predicted demands. The architecture is composed of Network Control and NFV Management. The Network Control uses ABNO principles to control the optimal network while the NFV Management coordinate the availability of virtual resources. In this way, the users can develop end to end services ensuring enough frequency slot widths. The feasibility of the framework is demonstrated with the implementation of a virtual Content Delivery Network (vCDN) for video streams and TV services.

3.4.3 Network Virtualization Services

The NFV approach offers several opportunities to service providers in the development of new services and applications [KT417c] , [tno17]. However, depending on the application, some NFV proposals can offer special advantages to developers in a specific area. Next, some additional NFV-based solutions for NF developers are described.

CloudNaaS [BASS11] is a novel framework that enables service providers the use of NFs and provide services in the infrastructure, such as isolation, QoS, custom addressing, among others. The design includes two main components: the cloud and network controller. The cloud controller coordinates the physical hosts and the corresponding virtual resources. For its part, the network controller is responsible for the virtual and

physical network devices. Similarly, CloudNaaS defines four principal operations. The first operation is the definition of requirements using a defined policy language. Then, the policies are translated to a communication matrix in order to find the optimal place to locate VMs. Then, the corresponding configuration rules are defined using a network language and, finally, the rules are installed in the network devices and VMs.

For its part, OpenADN [PJ12] proposes the use of SDN principles together with features of application SP (APs) to enable the provisioning of services and application in a distributed environment. The design is focused on cloud environments and includes four layers: virtualization layer to slice the network resources, NOS layer to control the OpenFlow network devices, a network level which invokes the ISPs services and a control level for the control of each ASP application. The implementation includes technologies such as OpenFlow, MPLS, slicing and cross-layer communication.

The Project VMware NSX [vS13] enables the onboarding of a VN as fast as a *Virtual Machine* (VM) in the provisioning of network virtualization services (e.g. compute or storage). For this purpose, NSX uses the Nicira network virtualization platform (NVP) to provide programability to virtual networks using SDN and adding a layer between the network and final hosts. The layers of this architecture is described as follows. The data plane includes a virtual switch (NSX vSwitch) which abstract the physical resources and connect with hypervisor. In this scenario, the NSX Edge acts as a Gateway between logical and physical network. Then, the NSX controller provides the control plane or SDN controller. Finally, the management plane implements a vSphere to enable the high level configuration and a cloud migration portal to control the migration and management tasks in virtual and cloud environments.

3.5 Summary

This chapter summarizes the NFV architecture. First, the virtualization in traditional architectures are analyzed. Then, the NFV principles are reviewed. Next, the NFV main implementation tools are reviewed. Similarly, the relevant NFV applications and use cases are presented

Chapter 4

5G Generation Mobile Network

This chapter reviews the main concepts related to 5G Networks, their requirements, ongoing work, KPIs, future trends and challenges and key-enabled technologies that leverage these kind of systems. This chapter is organized in 5 sections. Section 4.1 presents a general overview of 5G networks. Section 4.2 describes the 5G requirements and the fields where they can be applied. Section 4.3 discusses the different KPIs to measure the accomplishment of 5G requirements. In Section 4.4 the future trends and challenges are discussed. Lastly, Section 4.5 summarizes this chapter.

4.1 Overview

The emergence of a new business model and services (e-solutions, e-health, e-commerce, Voice over IP, streaming, among others) and the exponential growth in the information circulating on the Internet has brought unexpected challenges to the IT industry. The development of new mobile infrastructures, is focused on ensuring robustness, security, scalability and the fast deployment of applications through the customization of network behaviour. According to the Future Internet 2020 Report of the European Commission, the development of a new generation of networks takes an average of 10 years, this means that the fifth Generation Mobile Systems (5G) are coming soon [HNS+09].

5G must provide a flexible, reliable, secure, smart and high-performance environment to connect the digital society, while leveraging the competitiveness, faster innovation and standardization of new technologies. This network must embrace not only current services but also any kind of elements (IoT). These kind of networks will generate a significant impact not only on the societal but also on the operational field. On one hand, 5G must cover the necessities of smart cities, entertainment, public security, etc., providing a wide range of network services and applications [Nok14]. Users will expect enhanced QoE with minimal disruptions of the services, regardless of their location, the kind of device, or when the service is required. On the other hand, 5G will help to decrease the capital and operational expenditures (capex/opex) related to the deployment and management of new applications and infrastructures with substantially reduced service creation time [AIS+14].

Nowadays, the introduction of novel technologies is a time-consuming process due

the slow standardization process, manual service deployment or the semi-automated management tasks. At the same time, the *Average Revenue Per User* (ARPU) is continuously decreasing, while the demand on mobile traffic keeps growing. This causes a negative response by network operators to invest in new network hardware infrastructure. In order to lay the foundations of 5G Networks, three fields must be improved: Radio, Network and Operations and Management capabilities [EHE15].

- Radio Access capabilities leverage the spectrum optimization, enhance interference coordination mechanisms and support dynamic radio topologies through the exploitation of higher frequencies, enabling cost-effective dense deployments, intelligent and dynamic coordination of multi *Radio Access Technology* (RAT), as well as sharing resources, among others. Radio capabilities are intended to enable high data volumes, high mobility and spectrum efficiency.

- Network functionalities will enable the creation of an open environment in order to support several use cases in a cost-effective manner by means of the enhancement of user devices, minimizing the number of deployed entities and splitting the control and user plane functions (open its interfaces). These functionalities are also intended to ensure QoS levels.

- The operation and management capabilities are intended to simplify operations not only in network control tasks but also in the deployment of new services, without increasing the system complexity. This field also includes reactive and proactive mechanisms to enhance the decision-making in control and management operations based on network status and user profiles. This characteristic will enable the suited allocation of virtualized components, wherever they might be needed according the network status.

Radio and Network capabilities are topics well-studied in the literature [BHL+14], [GJ15], [BTAS14], [ABC+14]. In [GJ15], a detailed survey and ongoing projects related to 5G networks are presented. This work discusses some emerging technologies, such as massive *Multiple Input Multiple Output* (MIMO), cognitive radio, cloud technologies and *Device to Device Communication* (D2D) in order to tackle the following requirements: enhanced data rate, spectral efficiency, lower latency, deployment and management of ultra-dense networks. For their part, Boccardi et al. [BHL+14] describe five disruptive concepts that might impact on the development of 5G Radio requirements. They take into account the ability for devices to communicate between themselves (*Machine to Machine Communication* (M2M)), spectrum and resource optimization (massive MIMO and millimeter wave), the introduction of a device-centric concept and smarter devices (allowed to play an active role in the network). Regarding Network capabilities, one of the main challenges is to create an open, multi-tenant and service-oriented environment to support large amounts of traffic while covering different kinds of QoS levels and *Service Level Agreement* (SLA), in terms of latency, bandwidth or jitter. This environment will allow a flexible reconfiguration of network devices and programmability features based on the device-level, application, user and environment context [BTAS14]. Meanwhile, the

introduction of intelligence in 5G systems might enable the improvement of the resource use (spectrum, transmission power levels and other radio resources), cost-effective energy mechanisms and flexible cell management (different sizes) [DGK+13].

In order to tackle operation and management capabilities and enable ubiquitous connectivity, the research community proposes the introduction of some key technologies, such as SDN [KRV+15], NFV [ETS13a] Cloud Computing [ZCB10], Self-Organizing Network (*Self Organizing Networks* (SON)) [BGMB14] and Machine Learning [BAX+16]. SDN is based on the separation of the control plane from the data plane in traditional network devices. This decomposition allows the centralized control of the network with greater automation capacities and it simplifies the management process. For its part, NFV allows the implementation of traditional NF as virtualized instances, running in a generic hardware. The main advantage of NFV is its improved scalability capacity which, due to VNF, can be deployed anytime and anywhere in minutes, whereas previously it took more time (compared with traditional functions) [MSG+16]. From the technical point of view, SDN and NFV are complementary technologies, and together could facilitate configuration and network customization [CDLL15]. For their part, concepts such as Cloud Computing and SON allow the easy deployment of services (on-demand fashion) and enhanced traffic management based on intelligence decisions.

SDN, NFV, Cloud computing and SON are enablers that provide business agility and simplify the operation and management tasks. In contrast to traditional mobile systems, future networks will enable operators to control the traffic information (via SDN) in order to use only necessary network functions in a shared virtualized network (NFV and cloud computing). These technologies also allow the reduction of the complexity of planning, configuration and optimization tasks in the whole system, giving the capability to reuse existing infrastructures in a proactive way.

For its part, steps have been made by some stakeholders to cover future 5G needs [Nok14], [EHE15], [Moh15], [SC15], [Net14], [NEC15] such the definition of requirements and use cases, standardization and regulation activities, 5G testbeds, definition of KPIs, among others. In these activities have participated not only the academy and the main telecommunication service providers but also regulation bodies, *Small Medium Enterprises* (SMEs) and *Standards Developing Organizations* (SDOs). These organizations have helped to define a set of 5G requirements to address the needs of future mobile users, as is explained in the following sections.

4.2 5G Requirements

5G, moves towards bringing solutions to deploying faster networks, with hundreds of thousands of simultaneous connections and massive data transfer, while ensuring the quality of the new services. For this purpose, some requirements have been defined in six main dimensions [EHE15], as is shown in Figure 4.1.

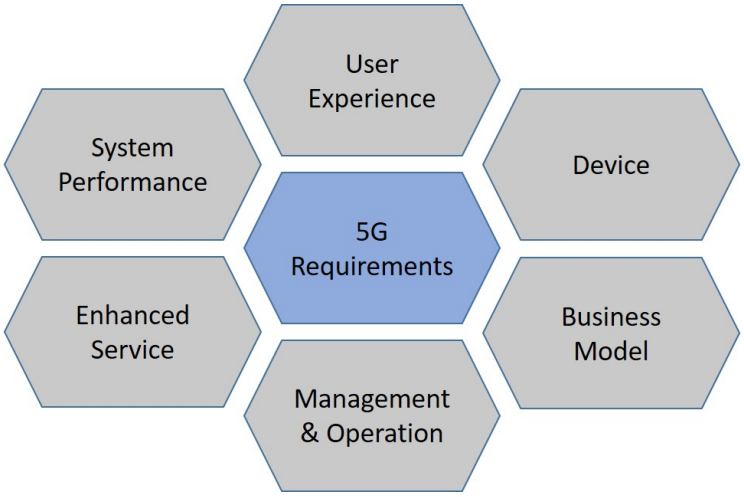


Figure 4.1: 5G requirements [EHE15]

4.2.1 User Experience

User Experience requirement focuses on maintaining and improving the user experience and their QoS levels across a broad range of mobility scenarios, in terms of bandwidth and delay. 5G users will benefit from consistent and high-quality service, regardless of network operator or their location with enhanced user experience compared with 4G systems even in critical situations such as very crowded areas, emergency events, network failures, etc. User experience includes high quality video services at HD or U-HD resolution [Nok14], [Net14] anytime and anywhere, enjoying a seamless experience.

4.2.2 System Performance

Currently, the spread of new services and devices, such as IoTs, autonomic vehicles, smart robots, virtual reality, etc, brings new requirements on system performance. 5G will be able to support the number of simultaneous connections per square kilometre, ultra-reliability, low end-to-end latency, which go beyond the traditional performance metrics, like capacity (data rate), coverage, the maximal mobility speed, and the number of active terminals per cell. As an example, the requirement on data rate will be substantially increased with capacities around 10 Gbps [Nok14], [EHE15], [Moh15], [SC15].

4.2.3 Devices Requirements

The main objective is to achieve high data rates, resources efficiency and signalling efficiency. New generation of devices like smartphones, robots, sensors, etc, will not only bring an increment of $x1000$ times in mobile traffic by 2020, but also impose a wide range of differentiated QoS requirements that will be meet by 5G systems [Moh15]. Device to device D2D communication, aggregation capabilities and smart devices with multiple bands and multiple modes will be also required in 5G infrastructures. Another important issue is to increase the battery life of current devices, for both smartphones and low cost sensors [EHE15]. For this purpose, 5G terminals must have a high degree

of programmability and configurability for both hardware and software components, in terms of transport protocol, terminal capabilities, access technology, etc. The operator will be able to monitor and analyse the network and service information in order to detect possible problems such as video drops or link failure, and then optimize and prevent harmful situations.

4.2.4 Enhanced Services

This requirement is intended to improve the user experience through the transparent connectivity, advanced location capabilities, high availability, reliability and resilience while providing high level of security in 5G environments. End devices will be able to connect to several RAT in order to enable transparent connectivity [CZA+15]. It takes into account the connection with a specific RAT or a combination of RATs and a seamless connection from both the user and the network perspective. For its part, 5G systems will be aware of contextual information such as the location in order to provide tracking of moving terminals. 5G will also enable extremely high network availability and reliability and self-healing capabilities to improve network resilience, especially in critical situations such as public safety or natural disasters. Furthermore, 5G devices will enjoy enhanced security services provided by robust authentication and user privacy.

4.2.5 New Business Models

In contrast to traditional mobile networks, 5G will enable operators to configure the data plane of network devices in order to control and customize the network behaviour without having architectural impact. In this way, the network and service provider will be able to introduce new business models while reducing the capital and operational expenditures. Thus, 5G may accelerate the TTM of new services and create independence from the hardware vendor [Net14]. 5G will also facilitate the evolution towards supporting enhanced levels of abstractions based on the separation between control and data plane. This approach enhances the coordination and isolation of access, configuration and management capabilities between service and network providers, similar to current services such as *Platform as a Service* (PaaS) or *Network as a Service* (NaaS). Furthermore, the 5G system may provide advanced network sharing schemes in order to enable flexibility in the provisioning of services. This can include spectrum sharing or spectrum selection [SC15].

4.2.6 Deployment, Operation and Management

The main idea behind this requirement is to facilitate the provisioning and control tasks when a service is required, while reducing the capital and operational cost and to accelerate the TTM of new services. 5G systems enable a cost efficiency approach to minimize the *Total Cost of Ownership* (TCO) not only in the deployment of 5G infrastructures but also in management tasks. In general terms, 5G systems will provide self-configuration, self-optimization and self-healing capabilities in order to reply to failures or unexpected problems. For this purpose, 5G networks shall be able to take decisions in a short time

and then apply countermeasures. For instance, configuration changes or deployment of new virtual functions [NCC+16]. In this way, 5G foster the resource and operation efficiency and the seamless innovation or upgrade. 5G will reduce complexity of planning, configuration and optimization tasks, giving the capability to reuse and smoothly upgrade existing network infrastructures. 5G design [Nok14], [Moh15] should provide reliability, not only on equipment uptime, but also in the provision of required data, regardless the specific technology or vendor. This characteristic is crucial on mobile communications for control and mission critical services. Additionally, 5G is expected to cover areas with low population density, where ultra-low cost deployments are required due to the very low ARPU.

4.3 5G Key Performance Indicators

It is expected that 5G requirements will be covered in 2020 as well as beyond 2020. In order to define a performance measurement that reflect the accomplishment of 5G requirements, several KPIs are defined (Figure 4.2). These KPIs take into account the vision of different organizations and their main objective is to improve current capacities such as lower latency, more capacity and mobility, higher reliability and availability [Nok14], [AIS+14], [EHE15], [Moh15], [SC15], [Net14], [NEC15]. KPIs have a significant impact on 5G infrastructures in three levels: societal, operational and innovation.

Regarding the societal level, 5G is expected to enable ubiquitous, robust and continuous service access for end users. 5G will also be able to provide reactive and proactive responses against network problems. For this purpose, low latency and advanced management capabilities are required. For instance, the introduction of intelligent mechanism aids to decide if specific physical device is not been used and therefore it will be shut down in order to reduce energy consumption. At operational level, the main purpose of 5G systems is to decrease the capital and operational costs when new service is required. 5G will reduce the creation and deployment lifecycle of new services. With regards to the innovation level, 5G fosters a wide range of opportunities in low dense areas, higher resource efficiency, advanced security, flexible transport network, extreme-reliable communications, among others [Moh15], [Net14].

In general terms, eight KPIs are common between the research community and several organizations: latency, peak data rate, mobility, number of connected devices, capacity, energy efficiency, location accuracy and operational cost (opex). The main idea behind these KPIs is to enhance the capabilities in each field, as is explained below.

- Peak Data Rate (10 Gbps)
  5G Networks will provide higher data rate than its predecessor LTE. It is expected to reach a peak data rate around $10 Gbps$ [Moh15], [SC15] regardless the user location or the number of connected devices.

- Latency (5 ms)
  5G expects to decrease the latency perceived by the user from the source to the destination (end-to-end latency). 5G also introduces the term “zero latency” which

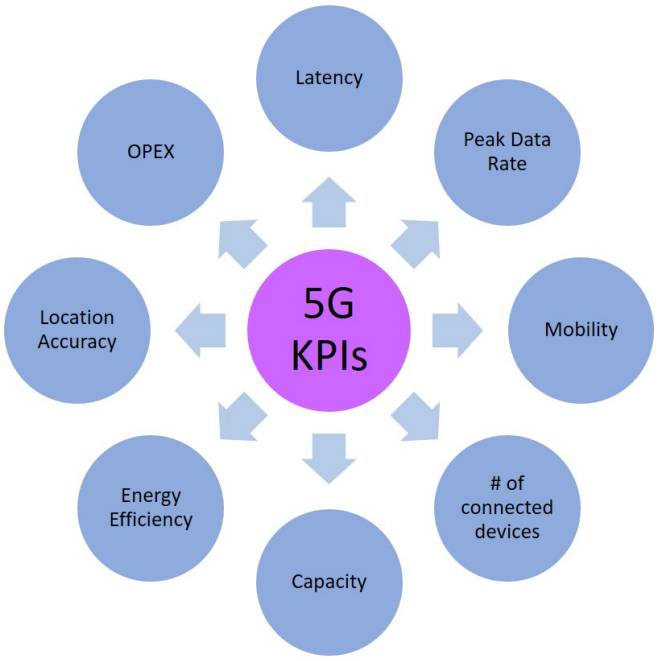


Figure 4.2: Summary of 5G key performance indicators

means there are no interruptions perceived by the user, taking into account the user expectations, the service quality and continuity. This KPI is a critical parameter in real time environments, such as public safety, real emergency systems, augmented reality, among others [OBB+14].

- Number of Connected devices (1 M/Km2)
  5G will be able to support 1000x number of connected devices compared to current infrastructures. 5G will take into account the concept of sensor networks or IoT, which envisage increasing the number of simultaneous connections in mobile networks and is expected to cover around 1M (devices) per square kilometre.

- Mobility (500 Km/h)
  This KPI enables end users to enjoy a seamless experience in fast-moving events like a train journey. This KPI is based on 5G user characteristics and their environment such as the diversity of mobile devices or the density of devices per square kilometre. 5G will also be able to support mobility speeds over 300 or 500 km/h [SC15]. These capabilities also require higher reliability and lower latency depends on the environment for instance broadband in dense areas, broadband to vehicles, etc [EHE15].

- Capacity (10 Tbps/Km2)
  5G will provide access to different kind of activities where high capacities are required such as dense areas with thousands of users are connected. 5G systems will be able to handle higher volume of traffic and variations, regardless of the connection density [EHE15].

- Energy efficiency (10%)

This KPI will cope the energy efficiency requirement on 5G networks. It is expected that energy consumption per service decreases by 10% compared to 2010 [Moh15]. It includes not only end devices but also the whole network components such as cloud, RAN and core elements [EHE15].

- Location accuracy (1 m)
  5G systems will be more efficient in terms of location accuracy (approximately 1m). This factor gains importance to deliver personalized services in real time services such as logistic transportation systems or road congestion information [NEC15].

- Service Creation Time (20%)
  This KPI is directly related to the reduction of operational expenditures (opex). 5G expect to reduce opex by 20% compared with current deployment, it means an approximate reduction of service creation time from 90 days to 90 minutes [Moh15].

Another transversal 5G KPIs are security robust, ubiquitous 5G access including in low dense areas, evolution of battery technology, reliability, improvements to facilitate dense deployments, among others [Nok14], [AIS+14], [EHE15], [Moh15], [SC15], [Net14], [NEC15]. In Table 4.1 the summary of 5G KPIs is presented, with their description and the expected value or percentage that they will meet.

Table 4.1: Summary of 5G key performance indicators

KPI	Description	Expected Value / Percentage	References
Latency	$\frac{1}{2}x$ end to end latency	$\leq 5ms$	[Nok14], [AIS+14], [EHE15], [Moh15], [SC15], [NEC15]
Peak Data Rate	$100x$ peak data rate	$\geq 10Gbps$	[Nok14], [AIS+14], [EHE15], [Moh15], [SC15], [NEC15]
Mobility	Mobility support to transport (vehicles)	$\geq 500Km/h$	[EHE15], [Moh15], [SC15]
Number of connected devices	$1000x$ number of connected devices	$\geq 1Mdevices/Km2$	[Nok14], [AIS+14], [EHE15], [Moh15], [SC15]
Capacity	Data volume density	$\geq 10Tbps/Km2$	[AIS+14], [EHE15], [Moh15], [NEC15]
Energy efficiency	10% lower energy consumption compared to traditional mobile networks (reduce $\frac{1}{10}x$)	10%	[AIS+14], [EHE15], [Moh15], [NEC15]
Location accuracy	Accuracy to determine the location of end devices	$\leq 1m$	[Moh15], [NEC15]
Opex	20% lower operational cost compared to traditional mobile networks (reduce $\frac{1}{6}x$)	20%	[AIS+14], [Moh15]

4.4 Future Trends and Challenges of 5G Networks

The current necessities address the direction of the business and the requirements of 5G Networks. It is expected that 5G networks will cover the increase of traffic volume by

means of improving spectrum utilization, enhanced energy efficiency mechanisms, resource virtualization, resource sharing, self-management and self-organization capabilities [ABC+14]. The concept of 5G envisages a broad range of opportunities in different fields. In other words, it will cover not only the traditional network fields but also other domains, such as e-health, energy efficiency, emergency services, public safety, IoT, machine-to-machine (M2M) communication, *Information Centric Networking* (ICN), among others.

The applicability of SDN, NFV, SON and cloud computing opens the door to facilitate the deployment and management of services in an open business environment, as is shown in Figure 4.3. On one hand, it presents a layered structure: infrastructure, virtualization, control and application layers, similar to the SDN approach. On the other hand, VNFs and NFV M&O modules are incorporated in order to control the NFVI. For its part, cloud technologies are present on the cloud computing layer and SON capacities will aid in the decision process in the control layer.

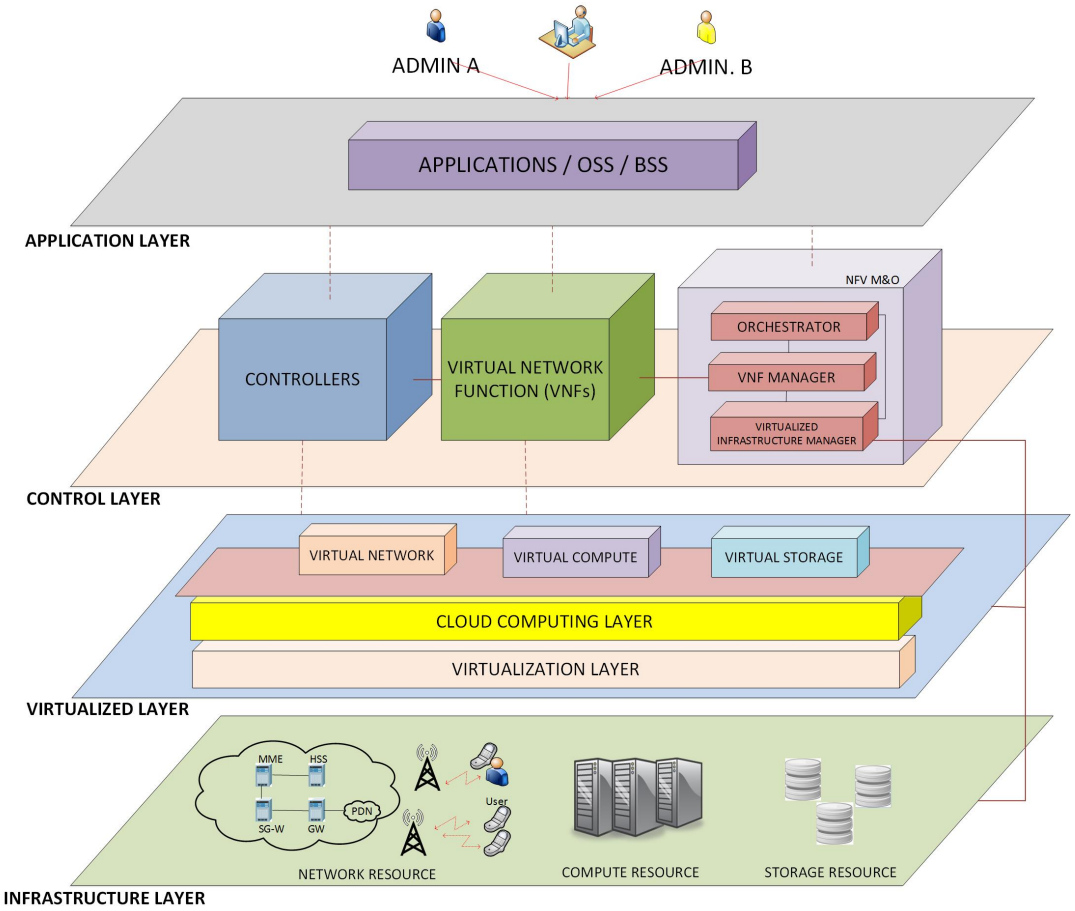


Figure 4.3: Future mobile network architecture

5G will incorporate all of these concepts or part of them. Despite the advantages of this proposal, there are some challenges that need to be overcome in order to successfully combine these technologies. Firstly, the unified definition and standardization in the separation of the data plane and control plane and the provision of virtualized instances

will enable the easy development and integration of the future network technologies. In addition, the complexity of the mobile network elements constitutes a big challenge by itself. At the same time, these kind of systems will require effective pricing schemes and business models with two objectives in mind: i) Customers pay only for the provisioned service and ii) stakeholders receive revenues according to their SLAs. Another important issue is how legacy networks will coexist with new systems, which is still a relatively unexplored field.

In the management and orchestration field, significant changes are required not only to improve the processing of data information but also to optimize the deployment and allocation of network resources. A unified management framework could allow enhanced traffic monitoring, provide self-management capabilities and network customization. A virtualized environment faces some issues, such as finding the best place to allocate virtual functions (operator infrastructure or cloud), migration and scheduling process. Mechanisms are also needed to provide load balancing, energy efficiency algorithms, inter-domain capabilities, among others. In parallel, all of these characteristics should be provided in a secure and trusted environment with enhanced capacities to recovery from failures. Moreover, the SDN centralized control or the dynamism of cloud computing are challenges that need to be covered. Table 4.2 shows the challenges and future trends that must be covered in order to fulfill the user needs of 5G networks.

Table 4.2: Current trends and challenges

Requirement	Challenge	Future Trends/Enabler Technologies
System Performance	Provide efficient mechanisms regarding to radio resource provisioning. Improving the capacity of radio resources. Provide super wide bandwidth. Better management of data traffic, interference and mobility levels.	Evolution of RATs. Decreasing the cell size. Millimeter-wave communication Intelligent resource allocation via SDN or SON.
Composite Wireless Infrastructures	The 5G device can choose the most appropriate wireless or mobile technology according their needs (Change between systems).	Enhancement of user devices (Muti-Band-Multi-Mode support). Introduction of intelligent mechanism and SDN control.
Facilitating very dense deployments (Hetnets)	Operators will must provide effective mechanisms to deploy cells of different sizes according to user needs.	Improving the resource capacity through decreasing the cell size. Introduction of intelligent and *Software Defined Radio* (SDR) concepts.
Flexible spectrum management	Improve the spectrum utilization in order to operate in some spectrum bands or channels, while reducing interferences.	Massive MIMO Mechanisms to use unused bands.
Native support D2D Communication	Deploy networks based on interconnected end user devices (machines, sensors,etc). The traffic will be properly assigned without cause congestions.	Introduction of Cognitive Intelligent mechanisms to exchange traffic between users. Smarter end-user devices.
Reduce Capex and Opex	Reduce the average service creation. Dynamic scalability and deployment of services and NFs, while reduce the complexity in planning and configuration tasks.	Resource sharing (Exploring Cloud-RAN, Cloud computing, NFV) Smarter allocation of functional mobile components (SDN, NFV).
Muti-tenancy and multi-service support	Service providers can control the resources deployed in a shared infrastructure (network, computing, mobile resources).	Cloud computing SDN NFV *Mobile Edge Computing* (MEC)
Open Environment	New applications and NFs could be deployed in an open environment, no matter the network hardware and technologies used by operators.	Standardization of SDN and NFV concepts. Introduction of SDR.
Energy efficiency operation	Saving energy per service provided. Nowadays, most of the energy consumption comes from RAN elements.	Introduction of intelligent and SON capabilities taking into account the device status.
Monitoring and Management	Provide self-management and self-optimization capabilities to 5G systems.	Automated management and monitoring functions (SDN, NFV). Takes decisions based on historical record of network status.
Ensuring QoS/QoE and SLA	A 5G user will be able to obtain enhanced services, regardless of the location or network technologies (compared with 4G systems), for several use cases such as emergency situations or network failures.	Enhanced mechanism to monitoring the network status (traffic optimization techniques) via SDN and intelligent mechanism. Automated network configuration to ensure the required need (SDN, NFV).
Charging and billing	Create different user profiles in order customers pay only the required service (pay-as-you-go), while operators bill the respective service.	Introduction of SDN and NFV concepts.

It is important to note the current efforts of initiatives such as 5G Americas [5G 17a], 5G-PPP or NGMN to develop 5G network. They promote not only SDN, NFV and cloud computing adoption but also the study of transversal concepts such as carrier aggregation, massive MIMO, Multi-RAT convergence, spectral and signalling efficiency, among others. It is imperative that telecommunication and network service providers find a consensus to develop solutions, architectures, technologies and standards for the next generation

of infrastructures. The communication paradigm of anytime, anyhow and anywhere will become a reality in the future society.

4.5 Summary

The present chapter reviews the vision of the 5G Generation Mobile Network. For this purpose, the main 5G requirements in terms of User Experience, System Performance, Device and Enhanced Services, New Business models and the Operation and Management are summarized. Then, the 5G KPIs are described. Finally, the future trends and challenges are analysed.

Chapter 5

Related Works

This chapter reviews the related works of Monitoring, QoS on SDN and Research Projects on 5G. This chapter is organized in 3 Sections. The Section 5.1 summarizes the advances on SDN Monitoring and QoS. The research projects on 5G are described in the Section 5.2. Lastly, the Section 5.3 summarizes this chapter.

5.1 Monitoring and QoS on SDN

The network monitoring solutions have been extensively studied in traditional IP networks. Depending on the strategy used to measure the values, the solutions can be classified as active or passive. In active methods, probe packets are included together with the normal traffic and sent through the network. Then, these probe packets are collected and analyzed to estimate values such as delay, end to end connection status, or round-trip time (e.g. ICMP packets). For its part, passive methods do not add additional packets to the network. The monitor observes the packets without influencing the network performance. This task is performed by agents in devices that observe the traffic (e.g. number of packets transported by a port) and send the information using a monitoring protocol. Protocols such as *Simple Network Management Protocol* (SNMP) [CFSD90] or *Network Configuration Protocol* (NETCONF) [EBSB11] are extensible used to transmit the information from agents that continuously reads the actual state of the network. Also, monitoring tools such as NetFlow [Cla04], sFlow [PL04] or jFlow [Mye99] apply statistical sampling to estimate flow based measurements. It is clear that the information obtained corresponds to network-layer measurements due the network having no access to users devices and the corresponding application level metrics.

Furthermore, the OpenFlow protocol enables switches to send information about the state of the switch through OpenFlow messages. This information can be used to estimate the actual situation of the network. In this context, the PayLess project [CBAB14] proposes a low cost efficient network statistics collection framework. The framework is designed on top of the OF-controller (NOX, Floodlight) and provides a northbound high-level RESTful API, as described in the Figure 5.1. The framework is composed of the following modules: Request Interpreter, Scheduler, Switch Selector, Aggregator & Data Store. The Request Interpreter translates the high level requirements requested

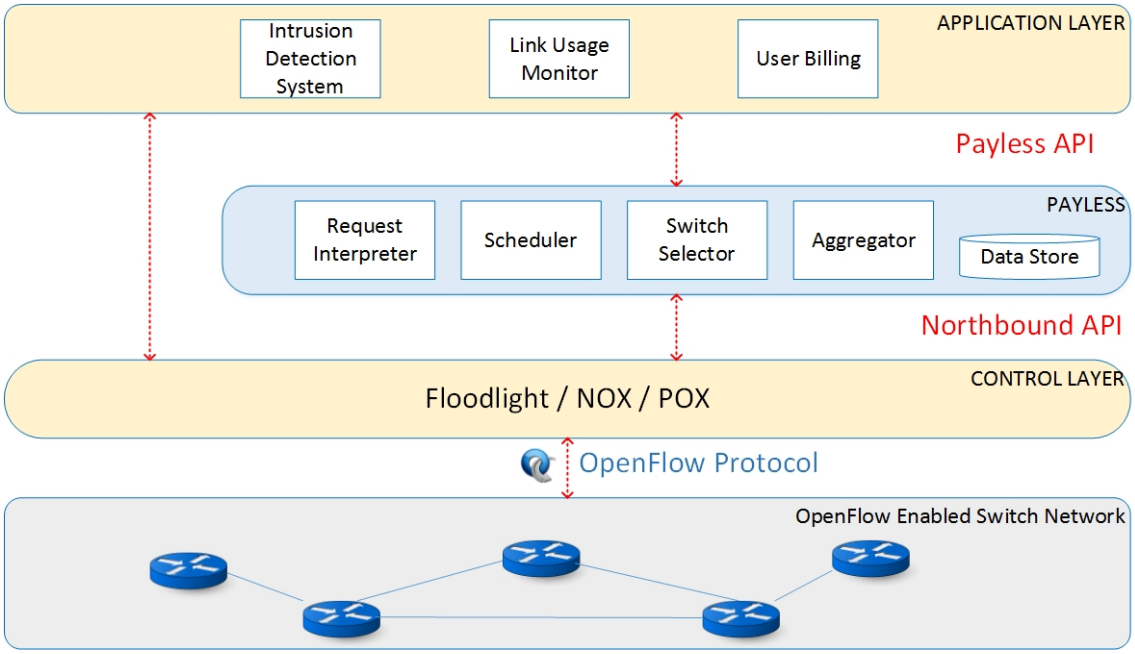


Figure 5.1: Payless architecture

by applications into network level primitives. The Scheduler coordinates the polling of switches for gathering statistics using the OpenFlow protocol. The type of requests (flow, queue and port statistics) depends on the applications. The Switch Selector selects the set of switches required to obtain the statistics as well as the time stamps. The Aggregator & Data Store collects and store the raw data.

Similarly, it provides aggregation levels to improve the quality and reduce the amount of information. This work also proposes an adaptive algorithm to request statistics and reduce overhead. The link utilization is measured with the information of flow removed and statistic request messages. The implementation of the algorithm in Floodlight controller and the measurement of link utilization in a link between switches demonstrate the effectiveness of the framework. The obtained utilization pattern very closely follows the pattern obtained in a traditional periodic polling and reduces the number of requests up to 50

The MonSamp [RSC14] proposes an SDN application for traffic sampling in order to extract a subset of the packets handled by the OpenFlow switch, but avoiding the over-utilized monitoring link. For this purpose, the SDN application aims to move sampling away from dedicated hardware to standard network hardware. For this purpose, MonSamp sends a copy of the traffic to a monitor agent. The monitor agent is composed of the collector and analyzer. The collector infers the current workload of the switches and sends this information to the MonSamp application. Then, MonSamp can reactively adjust the flow sampling and decide a reduction of the number of flows in order to avoid random drops. Similarly, if the links and collectors have available resources, the MonSamp application can dynamically increase the number of flows. These instructions are transformed into OpenFlow rules to be installed in the switches. For this purpose, the Pyretic language is used to avoid duplicity and guarantee the expected behaviour of

different applications. In a similar way, the information provided by the collector is sent to the analyzer. The results of analysis procedures is sent to the Traffic Engineering process and high level applications to take decisions.

The implementation of MonSamp is tested using a Mininet topology containing three nodes and an OpenvSwitch. The emulated network runs a real network trace composed of a PCAP 18 000 packets, distributed into 140 TCP and 53 flows (gateway of 40 students and researchers). The monitoring link is limited with a bandwidth of 1 Mbps. The testbed was tested with different average speed levels (0.7 Mpbs, 0.94 Mbps, 1.17 Mbps, 1.4 Mbps). The tests show that the monitoring of flows is possible and the impact of monitoring tasks does not compromise the normal behavior of the traffic.

OpenNetMon [vADK14] proposes a POX OpenFlow module that enables the monitoring of per-flow throughput, packet loss and delay. In this context, the number and type of active flows is determined by the difference between *packet in* and *flow removed* messages. Once the active flows are identified, the throughput is measured through the query of *flow statistics* messages. The query algorithm enables the change of sample rating depending of the stability of the flow. In the case of packet loss, OpenNetMon is calculated by polling flow statistics from the first and last switch of each path. The packet loss is measured with the difference between the increase of the packet counter on source switch and the packet counter of the destination switch. The delay algorithm uses the OpenFlow capabilities to inject traffic into the network. The additional probe packets follows the same path that the monitored flow. When the probe packet arrives into the destination switch, the controller compares the inter-arrival time and subtracting the estimated latency from the switch-to-controller delays. The implementation is developed through two modules: forwarding and monitoring. The forwarding enables the reservation and installation of paths and the monitoring performs the monitoring tasks. The results of experiments demonstrate the accuracy of throughput and delay measurements and a good estimation of service degradation in packet loss.

Furthermore, OpenTM [TGG10] proposes a traffic matrix estimator for OpenFlow networks in order to reduce the overhead while the system is performing monitoring tasks. For this purpose, OpenTM algorithm uses the router information available in the controller to select the switches to request statistics without compromising the accuracy of measurements. The work analyzes different selection strategies: querying the last switch, querying switches on the flow path uniformly at random, round-robin querying, non-uniform random querying and querying the least loaded switch. The implementation of OpenTM as a C++ application for NOX and the results of experiments demonstrate that the maximum difference between the best and worst querying schemes is about 2.3%. That means that any of these querying schemes can be used. However, the best performance method is the non-uniform random querying, as it tends to query switches closer to the destination with higher probability.

The solution presented in [STH14] proposes a diagnosis scheme to monitor the link performance in an OpenFlow network. For this purpose, the authors use a trace-route like measurement technique and a method to reduce the number of flow-entries. The scheme uses a single beacon and probe packets to cover all links of the network. The proposed

method is composed of three well defined steps: calculating measurement paths, setting measurement paths and aggregation of measurement flows. First, the measurement paths to comprehensively cover all links are calculated. Then, the OpenFlow protocol is used to configure the switches to detect the probe packets and follow the previously assigned path. When the probe packets returns to the beacon, the system is able to estimate the link performance metrics. Finally, multiple flow entries can be aggregated into a single entry by applying common header options to probing packets. The results of experiments using the Trema emulator confirm that the proposed method has considerable potential to efficiently measure the performance in all links.

The framework proposed in [SBJN16] presents a monitoring solution that overcome the under-utilization and over-utilization of a SDN controller. The proposed design is focused on the monitoring of the SDN control plane status in a distributed environment. For this purpose, the solution is composed of two elements: Collection Manager and Monitoring Agent. The Collection Manager collects the real-time statistics of the individual controllers belonging to the clustered environment. Similarly, it provides an aggregation service to calculate the overall load of entire control plane. The Monitoring Agent pushes the individual controller metrics to the centralized Collection Manager. The agent is composed of read plug-in to collect the metrics and write plug-in to send the information to the Collection Manager. The monitoring solution is implemented in ONOS controller. The experimental setup uses a cluster of 3 SDN Controllers and a benchmark device used to overload the control plane. The results of experiments demonstrate the over-utilization and under-utilization of individual controller that participates in ONOS cluster environment.

The solution proposed in [SOP+16] uses a cloud-based real-time logging analysis engine to provide SDN monitoring capabilities. The framework is composed of a Log Management Agent and the Analysis Platform. The Log Management Agent automates the data collection from the log files and send the information to the Analysis Platform. Then, the Analysis Platform analyzes the collected information and notifies the administrator about critical events, possible network attacks or reports the overall health of the network. The implementation is performed using POX controller and the experimental setup is composed of an emulated SDN network (mininet) and SDN controller (POX). The test results demonstrate that the framework is able to collect the logs provided by POX controller and OpenvSwitch and filter different metrics (load per switch, packet types, drop Packet In events, CPU usage).

Some initiatives have already analyzed the opportunities of SDN/OpenFlow in the development of QoS applications. In [ECT13], the authors propose a SDN Framework to adaptive video streaming enabling a dynamic QoS routing support. The proposed framework (Figure 5.2) is composed of several key functions: topology management, route management, route calculation, service management, SLA management and traffic policing. The Service Management function receives the QoS option from the network and reviews if the requested SLA can be performed. Once the service is accepted, the route calculation listens the start of the video, calculates the route and configures the flow tables of the switches. The route management guarantees the QoS and reactivates the route calculation if there is congestion in the network or a new route must be established.

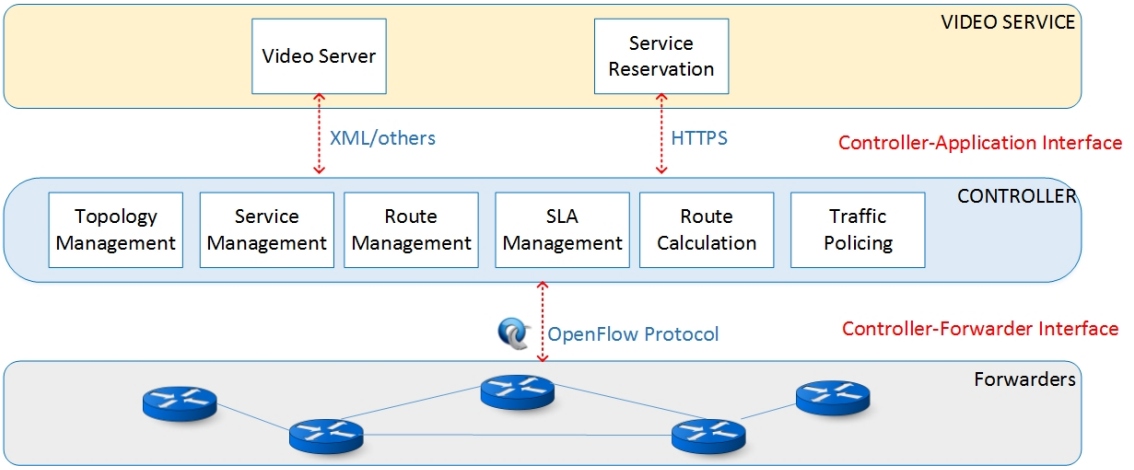


Figure 5.2: Adaptive video streaming framework

The traffic policing monitors the guarantee of SLA in the end points stated in the QoS contract. In this context, the framework offers three routing schemas: QoS level-1 (higher priority), QoS level-2 (low priority) and best effort engine. The routes are calculated based on the Constrained Shortest Path (CSP) problem and the used algorithm is the polynomial-time LARAC Algorithm [JSMR01]. The results of simulations demonstrate that the QoS routing attains in the range of 6-12 dB (QoS level-1) and 2dB (QoS level-2) overall PSNR improvement for real streaming video compared with best-effort engine.

The QoS Framework presented in [KSL+10] enables the automated fine-grained management of converged network fabric. The framework is composed of three well defined modules: Adaptive Aggregator, QoS Controller and Network Wide Optimization. The Adaptive Aggregator categorizes individual flow into groups (slices) and facilitates the allocation of resources based on the groups. The QoS Controller estimates the network state (e.g. network topology, active flows, and available resources) and uses these metrics for establish the QoS configurations. For its part, the Network Wide Optimization aims to maximize the satisfying of new flow requirements while minimizing the number of rejected flows. The results of simulations demonstrate that the flows with QoS priority suffer no packets loss and the TCP congestion is controlled.

Furthermore, in [BAE+13] the authors propose a Network Control Layer NCL integrating NaaS with SDN. The architecture is composed of the following components: SDNapp, SDN Controller and Monitor module. The SDNapp enables the control, management and configuration of the forwarding plane on demand. The main SDNapp functionalities include the resource allocator, provisioning, QoS tracker, application request engine and traffic engineering data base (TEDB). The SDN controller maintains the network rules and configures the flow tables of the OpenFlow switches following the instructions received by the SDNapps. Finally, the Monitor module is responsible collect statistics using OpenFlow counters and infers the network behavior. Similarly, it also receives the information from external monitoring tools. The framework prototype implementation is based on OpenNaaS tool.

The PolicyCop [BCAB13] proposes a framework to configure SLA and provide QoS.

This framework consists of a data, control and management plane. The data plane is composed of the data forwarding elements (e.g. OpenFlow switches). The control plane consist of a set of controller applications that implements the network functions. The main network functions include the admission control, routing, device tracker, statistics collector and Rule DB. The management plane is composed of policy validator and policy enforcer. The policy validator monitors the QoS of services and detects policy violations. In case of a policy violation, the policy enforcer re-provision the network resources. The authors uses Floodlight controller to implement the framework. The evaluation demonstrates that the policy adaptation algorithms can detect policy violations and quickly react to overcome the problem.

The work presented in [JKM+13] addresses the integration of OpenFlow with large scale WAN networks. The B4 Googles WAN network uses OpenFlow to simultaneously support standard routing protocols and centralized traffic engineering (TE). In this way, the SDN controller offers the following services: 1) control and adjudication of traffic demands during resource constraint, 2) leverage network capacity using multipath forwarding/tunneling according to the application priority, and 3) dynamic reallocation of bandwidth in case of failures or change in the demand requirements. The authors propose an architecture of three layers: switch hardware, site controller and global layer. The switch hardware layer is responsible of the forwarding of data. The site controller consist of Network Control Servers (NCS). The NCS is composed of OpenFlow Controllers (OFC) and Network Control Applications (NCA). The OFC receives the high level policies from NCA and establish the instructions to be configured in the forwarding devices. The global layer enables the locally centralized control of the NCA applications. It is mainly composed of an SDN gateway and a central TE server. The implementation of the architecture demonstrate that most B4 links can run at near 100% utilization and all links have an average of 70% utilization over long time periods. These results show an improvement of 2-3x efficiency in comparison with standard solutions.

Some authors consider the term QoE as an evolution of QoS. QoE takes into account the perceived acceptance of the user to a particular service. However, this term is in development and requires a solid concept. The project presented in [KSKD+12] improves the Quality of Experience using the path optimization engine. The framework consists of the QMOF located in the SDN application layer and PAF located in the SDN control layer. The QMOF participates in the session negotiation progress and determines if the service is feasible. When the service is accepted, the QMOF establish the optimal service configuration for the given service session. The configuration rules is composed of flow operating parameters (e.g. frame rate) or resource requirements (e.g. bit rate). The PAF exposes an API and receives the QMOF configuration rules. Then, the PAF optimizes the path routes through the distribution of the paths to each of the flows in a given configuration depending on the service requirements. The PAF uses OpenFlow messages to configure the network devices. In this context, the proposed framework aims to achieve the QoE of the users and perform optimized path assignment.

The OpenFlow-assisted QFF [GEB+13] analyzes and identifies the multimedia transmissions and the terminal device. QFF logically is composed of three parts: Input,

Intelligence and Output. The input is composed of the Network Inspector and Media Presentation Description (MPD). The Network Inspector monitors the number of devices in the network, the streaming bitrate requested by the devices and the available network capacity. The MPD Parser notifies the client video requests, such as video duration, bitrate, and the corresponding number and size of chunks. The intelligence is composed of Utility Functions and Optimization Function. The Utility Functions maps the bitrate of a video with the QoE perceived in the terminal device. The Optimization Function finds the optimal bitrate for each streaming video that results in an equivalent QoE level. Finally, the output is composed of the Flow Tables Manager (FTM) and the Dynamic Adaptive Streaming over HTTP (DASH) Plugin. The FTM installs the flow rules into the forwarding elements and the DASH Plugin coordinates the bitrate requests with the clients (terminal devices). The results of experiments demonstrate that QFF maintains the QoE in heterogeneous devices optimizing the data rates and providing network stability.

Table 5.1 summarizes the above contributions in terms of the framework structure, metrics and the evaluation process.

Tabla 5.1: Summary of related works on monitoring and QoS

Project	Components	Metrics	Implementation	Testbed	Results
Payless [CBAB14], Monitoring	Request Interpreter, Scheduler, Switch Selector, Aggregator, Data Store	Link Utilization	Floodlight, Mininet	Link meausrement on 3 level tree topology and UDP traffic sent by iperf	The utilization pattern follows the traditional approach, The reduction of request is up to 50%
MonSamp [RSC14], Monitoring	Collector, Analyzer	Bandwidth	Pyretic + POX, Mininet	Bandwidth on a topology of 3 nodes, 1 OVS and real network trace PCAP 18 000 packets (140 TCP, 53 flows)	The impact of bandwidth monitoring is low
OpenNetMon[vADK14], Monitoring	OpenNetMon controller module	Throughput, Packet loss, Delay	POX, 7 Intel Xeon Quad Core servers (1 Controller + 4 OpenvSwitch+2 Host)	Comparison between OpenNetMon metrics and static parameters (Tcpstat/Netem)	Throughput: deviation of 1.2%, Packet Loss: good estimation to detect service degradation, Delay: difference of 30% (unsuitable)
OpenTM [TGG10], Monitoring	OpenTM controller module	Average flow rate	NOX, HP DL320 G5p Servers, NEC IP8800/S3640 switces	Average Flow Rate of two pairs in a Linear Topology of 10 swicthes based on Iperf and Netem.	Average Flow Rate: deviation of 2.3%, Non-uniform random querying offers the best performance method
Active Performance [STH14], Monitoring	Controller module	Packet Delay, Loss Rate	Trema emulator, 2 Intel Xeon E5-2420 servers	1 Server with Trema emulates network and 1 Server acts as a beacon. Tcpdump captures the probe packets	Loss Rate: Estimation Error 10%, Packet Delay: Estimation Error 15%
Centralized controller [SBJN16], Monitoring	Collection Manager, Monitoring Agent	CPU, Memory, NetworkI/O load	ONOS, 4 Inter Xeon 2.3 Ghz server	Cbench generates traffic to overload 1 controller of a cluster of 3 controllers	The system shows single controller metrics and aggregated cluster results
Log Monitoring [SOP+16], Monitoring	Log Management agent, Analysis Platform	Switch load, Packet Types, CPU usage	POX, Mininet, Dell T5500 Server	Monitor of the status of switches and controller in a single, linear and tree topology	The information provided by log files are collected and the network health rating is displayed
Adaptive Video Fram.[ECT13], QoS	Topo., Route Cal/Man Service, SLA, Traf. Policing	QoS (PSNR)	LEMON emulator, GT-ITM emulator	PSNR comparison of MPEG video transmission over a test network of 300 nodes	QoS level 1: 6-12dB optimization, QoS level 2: 2dB
Scalable QoS[KSL+10], QoS, Monitoring	Adaptive Aggr., QoS Controller, Network Wide Optimization	Packet Loss, Throughput	NOX, HP ProCurve Switch, OpenvSwitch	QoS flows monitoring of a congested 2 level tree topology network	QoS flows do not suffer packet loss, The controller has a delay of 11.7 ms
PolicyCop [BCAB13], QoS	Admission Control, Rout., Dev. Tracker, Stat. Collector, Rule DB	Throughput	Floodlight, OpenvSwitch	Comparison of throughput response to link failures in a topology of 4 OVS	Detection of link failure and path rerouting
B4 [JKM+13], Monitoring	Switch Hardware, Site Controller, Global	Link Utilization	Onix, Googles WAN B4	Monitoring of Link Utilization of the B4 TE tasks	Most of B4 links run at near 100%, All links average of 70%
QoE Fairness [KSKD+12], QoS	Network Inspector, MPD Parser, Utility Opti. Function, Flow Tables Manager	Flow Bitrate	Switch TP-LINK WR1043ND, Pantou Openflow Implementation	QoE in terminals(smartphone, tablet, HDTV) in congested links in a Openflow network	QoE stable in heterogeneous devices, Network stability

5.2 Research Projects on 5G

5G Networks require customizable, efficient and scalable network infrastructures in order to meet the new user needs and the exponentially-increasing traffic demands, while decreasing the capital and operational expenditures. The SDN concept has been introduced in a broad range of fields, such as QoS, data centers, mobile and optical networks, security, network virtualization, among others [HHB14]. As an instance, Google was one of the first enterprises to incorporate the SDN concept to communicate their internal Datacenter-WAN. Furthermore, there are some projects that allow SDN experimentation by offering scalable testbed infrastructures with research purposes, such as Geant, GENI, Ofelia, Felix, among others [SNC+14].

In particular, the integration of SDN or NFV with mobile networks includes the deployment of virtualized base stations and core components LTE [PWH13], energy efficiency experimentation on WiFi networks, the optimization of very dense and heterogeneous wireless networks [cro17], etc. The next generation of mobile networks could take advantage of the combination of key-enabled technologies to enhance the following areas: (i) the development of radio access (high speed, spectrum efficiency, high mobility, high availability); (ii) improvements in core networks (QoS support, aggregated processes, network slicing, cloud deployment) and (iii) the management and orchestration process (customization of user needs, dynamic allocation of resources, energy efficiency mechanism, manage a big amount of data) [BHL+14] [DGK+13] [AHGZ16].

Different standard organizations leverage the adoption of SDN and NFV concepts in their infrastructures. These organizations have presented the challenges, KPIs and possible use cases in order to cover the above-mentioned areas. As an instance, ONF [ONF17] promotes the adoption of SDN and defines a wide range of use cases, such as inter-cell interference management, virtual customer edge, network virtualization or data center optimization. Meanwhile, NFV is an initiative of ETSI and telecommunication providers, which proposes the virtualization of the traditional network functions. ETSI-NFV defines nine general use cases [ETS], such as NFVIaaS, VNF-FG, etc. In the scope of mobile networks, NFV promotes the virtualization of Mobile Core Networks and *IP Multimedia Subsystem* (IMS), the virtualization of a mobile base station, the virtualization of the home environment and the virtualization of Content Delivery Networks (CDNs). In the meantime, some open source projects led by the research community have emerged to provide an open environment to test with SDN, NFV and cloud computing, such as OpenNFV (SDN and NFV) [KT217], Floodlight (OpenFlow and OpenStack support) [flo17], OpenDaylight (SDN, NFV and OpenStack) [MVTG14].

With regard to mobile networks, industry manufacturers, telecommunication operators, and related stakeholders are working on the definition of requirements, standardization, regulation and development of future mobile systems, such as 5G-PPP (5G Infrastructure Public Private Partnership) and Next Generation Mobile Network initiative (NGMN). The 5G-PPP [Moh15] proposes solutions, standards and infrastructures to allow for the ubiquitous 5G communication. For its part, the NGMN [EHE15] will expect to provide 5G solutions by 2020, within eight general use cases:

broadband access in dense areas, broadband access everywhere, high user mobility, massive Internet of Things, extreme real-time communication, lifeline communication, ultra-reliable communication and broadcast-like services. The most outstanding efforts have been made in the 5G research field. A wide range of projects or initiatives will expect to cover the needs of future mobile users. These worldwide initiatives encompass global regions of Asia, Europe and the Americas.

With the aim of promoting the adoption of 5G in Asia, China has launched the IMT-2020 promotion group [IMT17], which manages five working groups: Requirements, Technology, Spectrum, *Intellectual Property Right* (IPR) and Standardization. This is the most important promotion platform related with research and international cooperation purposes. Similarly, coordinated efforts in the 5G area have been launched in South Korea and Japan, the former with the 5G Forum [5G 17b] and the latter with the *Fifth Generation Mobile Communications Promotion Forum* (5GMF) [5GM17]. Both are conducting research projects involving active participants from the government, industry, and academia, in order to facilitate the development of 5G.

Significant efforts have been made in Europe under the support of the European Union Framework Project 7 (FP7) and the Horizon 2020 programmes [Moh15]. On the one hand, FP7 has launched 5G research projects such as METIS, *Mobile Cloud Network* (MCN), CONTENT, T-NOVA, UNIFY, CROWD, etc. On the other hand, Horizon 2020 has financed several research projects (considered by 5G-PPP as Phase 1 Projects) such as 5G-NORMA, METIS II, CHARISMA, SONATA, FLEX5GWARE, SELFNET, among others. In the FP7 context, Mobile and wireless communications Enablers for the Twenty-twenty Information Society Project (METIS) [OBB+14] lays the foundations of 5G networks and promotes the general agreement to design this mobile environment. The first phase of this project (METIS I) includes five big scenarios (amazingly fast, great Services in a crowd, ubiquitous things communicating, best experience follows you, super real-time and reliable connections) in a use-case driven approach.

The initiative Mobile Cloud Computing (MCN) [mcn17] provides mobile services by means of the combination of three components: mobile network, compute, and storage resources. MCN defines a wide range of use cases, such as Radio Access Network (RAN) on Demand, Mobile Virtual Resources on Demand, Machine Type Communication on Demand, SDN or virtualized Evolved Packet Core (EPC), to mention a few. In the same way, the CONTENT project [kt317a] proposes a network infrastructure that enables end-to-end cloud and mobile services. This project provides a virtualized infrastructure based on LTE, WIFI and optical metro networks and introduces the SDN concept in their deployment. CONTENT presents two general use cases: Infrastructure and network sharing (created logical resources) and cloud service provisioning on top of virtual infrastructures (end-to-end).

The integration of SDN with NFV is proposed in T-NOVA [tno17] and UNIFY [uni17] projects. On one hand, T-NOVA provides a framework to deploy NFVs over network infrastructures. The innovation of this project consists of their NFV Apps marketplace, which enables the easy creation, deployment and management of virtual network appliances in a standardized environment. T-NOVA proposes three general

scenarios: High-Level Scenario, T-NOVA NFVs, and VNF Chaining. On the other hand, the UNIFY project takes advantage of cloud computing and the virtualization concept to provide a novel network architecture with optimized data traffic flows and the dynamic placement of networking, computer and storage components. This project presents eleven use cases, organized around the following domains: Infrastructure Virtualization, Flexible Service Chaining and Network Service Chain Invocation for Providers.

In the area of SDN and SON, CROWD [cro17] includes these technologies to enhance the coordination process between radio base stations in very dense and heterogeneous wireless networks (Dense Nets). This project allows the network cooperation, the dynamic network configuration, dynamic backhaul reconfiguration, energy optimization, etc. CROWD also presents fifteen use cases divided into two big scenarios: self-optimizing dense networks and Optimized mobility in dense radio access networks.

As part of the Horizon 2020 programme, 5G-NORMA [nor17] is a research project which aimed to provision an adaptive and open 5G infrastructure with capabilities to service customization, enhanced performance and security. To this purpose, this project introduces adaptability capacity to allocate mobile network functions in the most appropriate location and in a short time. Likewise, METIS II [kt317b] presents a novel 5G RAN design, introducing a protocol stack architecture intended to provide a seamless integration of 5G radio technologies. The innovations of METIS II are focused on the spectrum management, air interfaces harmonization, resource management and a common control and user plane framework. The integration of them will support regulatory and standardization bodies.

Other ongoing H2020 projects that combine SDN and NFV technologies are CHARISMA [KT417a] and SONATA [KT417c]. CHARISMA will enable the deployment of an intelligent Cloud Radio Access Network (C-RAN) and virtualized Customer Premise Equipment (CPE). SONATA will support network function chaining and an enhanced orchestration process in order to allow service customization.

The provision of innovative hardware and software platforms to support 5G infrastructures is proposed in FLEX5GWARE [KT417b]. This project attempts to develop and prototype key components of 5G networks in the hardware and software domains. The main objective of this project is to deliver a highly reconfigurable hardware platform together with a well suited software platform, over which network elements and devices can be deployed following a modular, efficient and scalable approach. Several components must be deployed as 5G enablers, such as MIMO emulators, high-speed broadband converters, Filter Bank Multi-Carrier (FBMC) transceivers, Low-Density Parity Check (LDPC) codes, etc., with suitable interfaces to allow flexible software-based management schemes.

The integration of SDN, SON, NFV and Artificial Intelligence is encompassed by the SELFNET [NCC+16] [sel17] project, which introduces intelligent, self-organizing and autonomic capacities to 5G networks, taking advantage not only of SDN and SON but also NFV and Cloud Computing. This project will provide a scalable, extensible and smart architecture to foster innovation and decrease capital and operational expenditures derived from network management tasks. Moreover, SELFNET introduces the SON concept to facilitate the automatic management of network infrastructures. SON solutions are

typically classified into three domains: self-protection, self-optimization and self-healing, which are the use cases proposed by SELFNET. Likewise, the COGNET Project [XAY+16] proposes the introduction of machine learning, SDN and NFV in order to enhance monitoring tasks and autonomic network management. COGNET predicts the resource demand requirements and then changes its own configuration based on the network analysis (prediction, frauds, detecting error and security conditions).

Table 5.2 shows the current European use-case driven projects that tackle different 5G requirements, through a combination of SDN, NFV, SON and cloud computing concepts. All of these projects take into account SDN in different domains, such as e-health services, security, service chaining, multimedia optimization, etc.

Last, but not least, it is worth mentioning the research efforts in the Americas, where a group of telecommunication service providers and manufacturers created the 5G Americas [5G 17a], an organization intended to foster the development of LTE wireless technology leveraging the adoption of 5G in the North and South Americas society. At the same time, several activities have been conducted by academia. For instance, the *Berkeley Wireless Research Center* (BWRC) involves university, industry, government and other research stakeholders focused on exploring innovations in wireless communication systems based on radio frequency and millimeter wave technologies, which are its main challenge to develop reconfigurable radio architectures. Likewise, the *Broadband Wireless Access and Applications Center* (BWAC) involves around fifty research centers with the aim to collaborate with the industry in the creation of innovative and scalable wireless networks.

Table 5.2: Research projects in mobile networks

Project Name	Related Technologies	Main Objective	Scenarios/Use Cases
MCN [mcn17]	SDN, Cloud Computing	Enhanced traffic processing by means of the separation between radio hardware and packet forwardinghardware.	Cloud Computing for Mobile Network Operations, End-To-End Mobile Cloud
T-NOVA [tno17]	SDN, NFV	Design and implementation of an integrated architecture for the automated provision and management of VNFinfrastructures.	High-Level Scenario, NFVs, Service chaining
UNIFY [uni17]	SDN, NFV	The development of an automated and dynamic service provision platform, based on a service chaining architecture	Infrastructure Virtualization, Flexible Service Chaining, Network Service Chain Invocation for Providers
CROWD [cro17]	SDN, SON	The creation of technologies to support dynamic network functionality configuration and fine, on-demand, capacity tuning.	General scenario
5G-NORMA [nor17]	SDN, NFV	The development of an adaptive, customizable, secure and efficient mobile network architecture to deal with complex traffic demandfluctuations.	Multi-service, Multi-tenancy
CHARISMA [KT417a]	SDN, NFV	The creation of an intelligent and hierarchical routing and paravirtualized architecture to enhance end-to-end services.	General scenario
SELFNET [sel17]	SDN, NFV, SON, Cloud	The design and implementation of an autonomic network management framework to achieve self-organizing capabilities in managing 5G network infrastructures, leveraging an improvement in the overall userexperience.	Self-healing, Self-protection, Self-optimization
COGNET [XAY+16]	SDN, NFV, Machine Learning	Dynamic adaptation of the network resources (virtual network functions), while minimizing performance degradations and fulfill SLA requirements.	Situational Context, Just-in-time Services, User-Centric Services, Optimized Services, SLA Enforcement, Collaborative Resource Manage

5.3 Summary

This chapter presents the advances on SDN and 5G. In particular, the Monitoring and QoS optimizations for SDN networks are described. Similarly, the research projects focused on the development of 5G technologies are outlined.

Chapter 6

SELFNET SDN/NFV Self-Organized Networks

This chapter reviews the main advances of SDN/NFV-based self-management networks. Similarly, the Self-Organized Network Management in Virtualized and Software Defined Networks Project SELFNET is described. In the same way, each component of the framework is described.

This chapter is organized in 10 Sections. Section 6.1 introduces this chapter. Section 6.2 reviews the network management with SDN/NFV. Section 6.3 presents the self-organized network management framework for SDN/NFV. The next sections gives details of the layers and sublayers of SELFNET. Section 6.4 presents the infrastructure layer. Section 6.5 discusses the data network layer and the Section 6.6 presents the SON control layer. Section 6.7 reviews the the SON autonomic layer. The Section 6.8 studies the NFV orchestration and management layer. The Section 6.9 presents the SON access layer. Lastly, section 6.10 summarizes this chapter.

6.1 Introduction

The management and customization of network services have been limited by the rigidity of traditional network architectures and the increasing of both capital and operational expenditures. Actually, the resolution of common traditional network problems, such as link failures, security attack, QoS or QoE degradation, bottlenecks, among others, requires the direct involvement of network operators. The manual re-configuration of the equipments or even the installation of new equipment (router, NATs, firewalls) compromises the normal operation of the network and causes the disruption of the SLAs. Similarly, the creation of innovative value-added services is limited by the closed and proprietary hardware/software, and in some cases, all infrastructure may belong to the same provider.

Those limitations makes traditional network architectures unfeasible to meet the requirements of todays users, enterprises, and carriers. The solution proposed to solve these challenges is following the advances reached by computing, where developers can create their own applications using a high level programming language. The programs can

be executed in several equipments thanks to the abstractions of resources provided by the Operating Systems. In this context, the SDN and NFV appears as a promising strategy to reach those objectives. SDN proposes the decoupling of data and control planes in network devices enabling their independent development and evolution and a centralized view of the network. NFV promotes the migration from typical network equipments (DPI, firewall, load balancers) to a software package or NF that can be instantiated in a virtualized infrastructure. Both architectures are complementary and potentially could be integrated to provide an open network environment for developers.

Furthermore, the exponential growth of mobile devices and content together with the advent of cloud services bring additional challenges to operators and service providers. A radical decrease of integrated network management operations without negatively affecting the QoS/QoE and security is required. Similarly, a new model that integrates the access and management of mobile resources is promoted. The future 5G networks are expected to expose not only typical mobile broadband but also a heterogeneous, simplified and unified control. The network management expenditures can be reduced through automation of operations. In this context, a scalable management framework that includes data mining, pattern recognition, learning algorithms to reduce operation expenditures is challenging.

The SELFNET [sel17] uses SDN/NFV principles to provide smart autonomic management of network functions in order to resolve network problems or improve the QoS. SELFNET integrates the self-management paradigm with the use of data mining, learning algorithms, pattern recognition to identify the network behaviour and including 5G mobile architectures. For this purpose, the SELFNET architecture is composed of well defined layers: Infrastructure, Virtualized Network, SON Control, SON Autonomic and Access Layer. Within SON Autonomic Layer, the Monitor and Analyzer sublayer is one of the most challenging operations. Monitor and Analyzer, on its turn, is divided in three modules: Monitoring and Discovery, Aggregation and Correlation, and Analyzer.

6.2 Network Management with SDN/NFV

The SDN and NFV principles offer several advantages over traditional network management architectures. Several consortiums formed by operators, academia, service providers have focused their effort on development of novel management architectures over virtualized environments. In Table 6.1, relevant SDN/NFV-based management projects are described.

6.3 SELFNET Self-Organized Network Management for SDN/NFV

The SELFNET H2020 project will design and implement an autonomic network management framework to provide SON capabilities in new 5G mobile network infrastructures. By automatically detecting and mitigating a range of common network problems, currently manually addressed by network administrators, SELFNET will

Table 6.1: Network management projects based on SDN/NFV

Project	Domain	Description	Application Scenario
CROWD [cro17]	SDN, SON	The project aims to bringing density proportional capacity in heterogeneous wireless access networks. Similarly, it focuses on guaranteeing mobile users QoE, optimizing MAC mechanisms and proportional energy consumption. In this way, it enhance the traffic management in dense wireless networks	Traffic Management
5G-NORMA [nor17]	SDN, NFV	The project focuses on providing adaptability of a resource in efficient way. The framework handle fluctuations in traffic demand resulting from heterogeneous and dynamically changing service portfolio. The novel network functions offer resource-efficient support of varying scenarios and help to increase energy-efficiency	Multi-service scenario, Multi-tenancy scenario
MCN [mcn17]	SDN	The project focuses on the enhancement of traffic traffic processing by means the separation between radio hardware and packet forwarding hardware	SDN environment
UNIFY [uni17]	SDN, NFV	The project aims to develop an automated, dynamic service creation platform through the creation of a service model and service reaction language. It will enable the dynamic and automatic placement of networking, computing and storage components across the infrastructure. Similarly, the orchestrator will include optimization algorithms to ensure optimal placement of elementary service components across the infrastructure	Infrastructure Virtualization, Flexible Service Chaining, Network Service Chain Invocation for Providers
T-NOVA [tno17]	SDN, NFV	The project focuses on the deployment on *Network Functions-as-a-Service* (NFaaS) over virtualised Network/IT infrastructures. For this purpose, it design and implement a managementorchestration platform for the automated provision, configuration, monitoring and optimization of virtualized resources. Moreover, the SDN is also used for efficient management of the network infrastructure	High-Level Scenario, VNF Chaining Scenario

provide a framework that can significantly reduce operational costs and consequently improve the user experience [sel17] [NCC+16].

By exploring the integration of novel technologies such as SDN, NFV, SON, Cloud computing, Artificial Intelligence, QoS/QoE and next generation of networking concepts. SELFNET will provide a scalable, extensible and smart network management system. The framework will assist network operators to perform key management tasks such as automatic deployment of SDN/NFV applications that provide automated network monitoring and autonomic network maintenance delivered by defining high-level tactical measures and enabling autonomic corrective and preventive actions to mitigate existing or potential network problems. SELFNET will address three major network management concerns by providing self-protection capabilities against distributed network attacks, self-healing capabilities against network failures, and self-optimization features to improve dynamically the performance of the network and the QoE of the users. The facilities provided by SELFNET will provide the foundations for delivering some of the 5G requirements defined by 5G-PPP consortium.

In this context, Figure 6.1 illustrates the architecture of the SELFNET framework. The architecture is based on five differentiated layers with the following logical scopes:

Infrastructure Layer, Data Network Layer, SON Control Layer, SON Autonomic Layer, NFV Orchestration & Management Layer, SON Access Layer. In the following sections, each layer will be described.

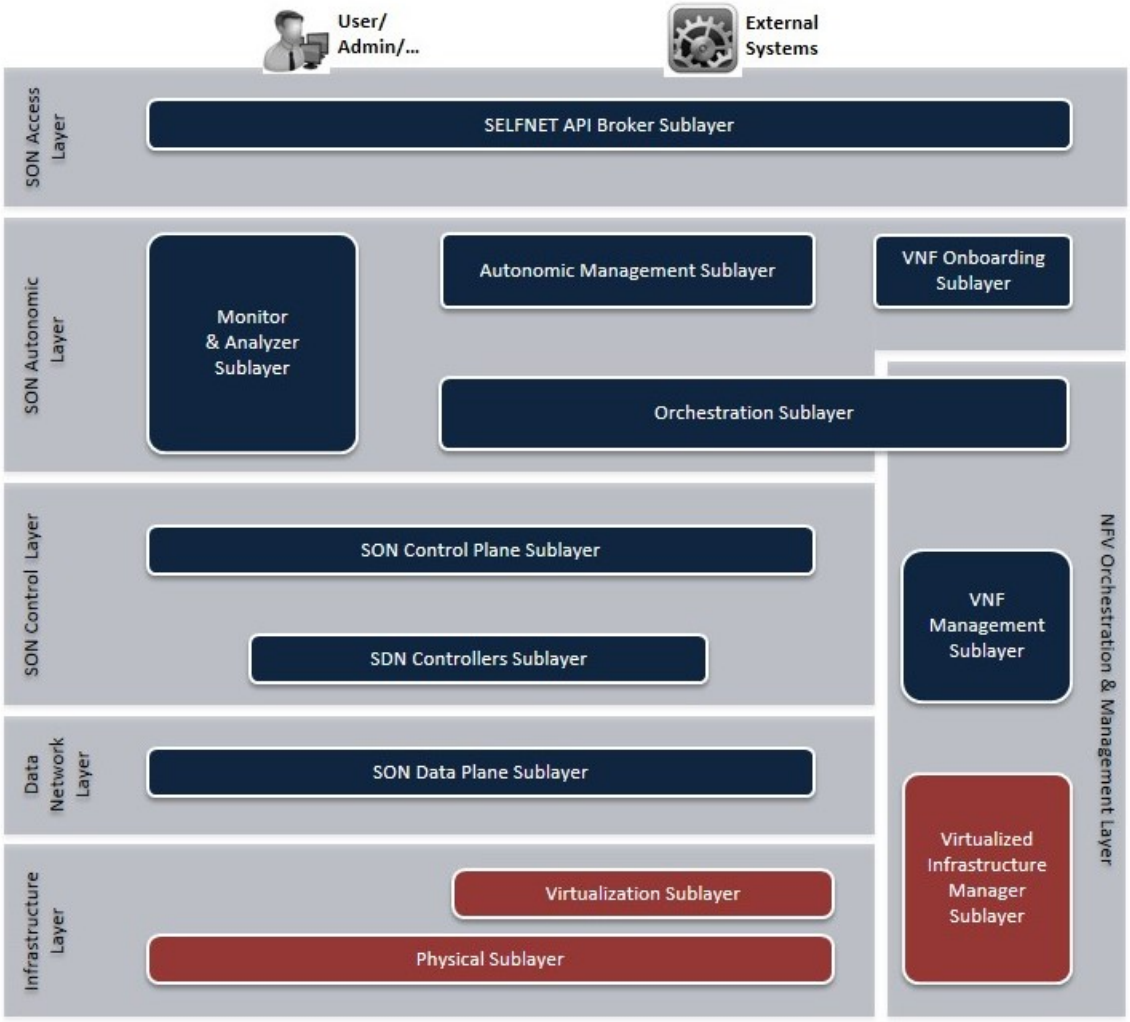


Figure 6.1: SELFNET architecture overview [NCC+16]

6.4 Infrastructure Layer

This layer provides the resources required for the instantiation of virtual functions (Compute, Network and Storage) and supports the mechanisms for that instantiation. It represents the NFVI as defined by the ETSI NFV terminology [ETS13a]. In order to achieve its functionality, two sublayers will be defined: Physical Sublayer and Virtualization Sublayer.

6.4.1 The Physical Sublayer

The Physical Sublayer includes the physical resources required to provide computation, networking and storage capabilities over bare metal. Since SELFNET is designed to

include 5G networks, the physical elements follows a mobile edge architecture in which operators can deploy the operational and management services. The MEC proposed by ETSI [PNC+14] is depicted in Figure 6.2. It proposes the edge nodes geographically separated from the data centre. In this way, the services could be deployed close to the user if operational or management services require high performance. Moreover, the integration of edge deployments (e.g. C-RAN) within the MEC discontinue the typical rigidity and allows the customization of services. Similarly, it is considered that the connectivity between the elements enables virtualization capabilities following the advances on 5G.

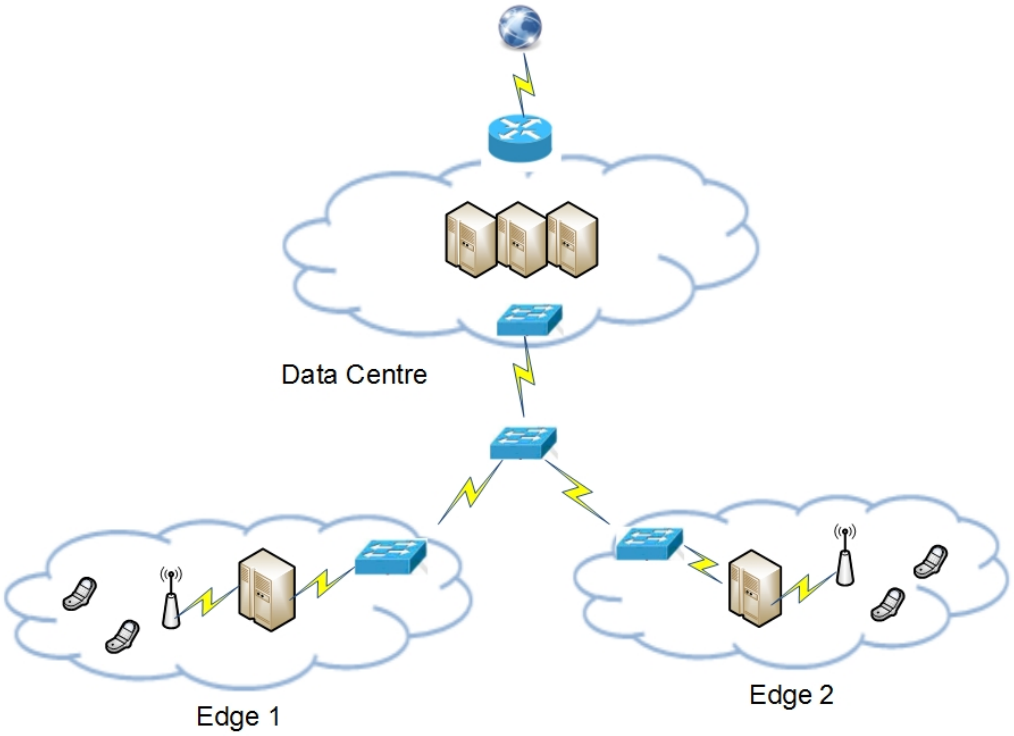


Figure 6.2: Physical layer of MEC infrastructures for 5G architectures

6.4.2 The Virtualization Sublayer

The virtualization sublayer enables the sharing of the available resources between different users or services. It offers several advantages, such as the isolation, reliability, adaptability and control of resources. However, the main disadvantage can include the performance penalti of the virtualization related tasks. In this context, recent advances on virtualization technologies have reached the expectations on 5G infrastructures on the performance of virtualized workload with intensive Input/Output (I/O) [AGHE+15]. In other words, the performance penalti of using virtualization can be considered as negligible for modern devices. In SELFNET, the virtualization sublayer includes the use of virtual switches used to connect virtual Machines allocated on the physical resources.

6.5 Data Network Layer

In this layer, the different functionalities of the networks functions are located and interconnected in a designed topology. The NFs include the instances required for the normal operation of the virtual infrastructure and those created by SELFNET as part of the SON functionalities. Because of the edge and centre location are fully virtualized, the NFs can be dynamically allocated in both locations.

The Data Network Layer also provides multi-tenancy support. Multitenancy enables the sharing of resources among different tenants, each with their own administrative domain and business requirements. In 5G architectures, the resources can be acquired by a telco Alliance and the resources are shared according to their needs. In this context, a specific telco administrator is not able to administer resources of other telco or intercept traffic provided by other traffic.

6.6 SON Control Layer

This layer includes the elements responsible of collecting data from different virtualized sources (SON Sensors) and the functions that execute actions into the network (SON Actuators). The SON Sensors and SON Actuators are controlled by the SON Autonomic Layer, which provides network intelligence. Similarly, the SON Control Layer deal with the control plane in SDN architectures. In other words, it translate an autonomic network-wide policies into specific network elements configurations.

6.6.1 SDN Controller Sublayer

The SDN Controller Sublayer implements a logically centralized controller (e.g. SDN control plane). It provides the governance of the network elements and controls the network functions. The SDN controller uses the information about the actual network behaviour and then can enforce the rules configured in the network elements. In this way, the traffic passing through such network elements can be dynamically modified. The interface between the controller and the network elements depends on the network device capabilities. For instance, protocols such as OpenFlow, OVS, OFConfig, NETCONF can be supported by an SDN Controller. The different SDN controller services are deployed by a set of SDN controller Applications or SDN-Apps. Examples of SDN-Apps includes routing protocol, forwarding protocol, filtering protocol or tagging protocol. In northbound, the SDN Controller provides an API to enable the remote management, configuration and monitoring of the controller behaviour.

6.6.2 SON Control Plane Sublayer

The SON Control Plane instantiate the different network functions NF running in the virtualized infrastructure. In SELFNET architecture, which aims to provide self-organized capabilities, there are two types of NFs: SON Sensors and SON Actuators.

- **SON Sensors**. It collects data related to network activities. The collected information includes metrics related to global traffic (e.g. link status, bandwidth) or specific metrics (e.g. DPI, QoS on a video streaming related to a specific data flow). The operators and service provider can develop different sensors according to their needs.

- **SON Actuators**. It executes a specific set of actions on the traffic circulating in the network. The actions depends on the application developed by the service providers. For instance, if the system detects a DDoS attack, a SON actuator can automatically block the specific attack source. For its part, if the system detects a QoS degradation, another SON Actuator can optimize the network flow increasing the priority or bandwidth.

6.7 SON Autonomic Layer

This layer in responsible to provide the network intelligence. The information collected from sensors is used to diagnose the network situation. Then, the actions to accomplish the systems goals are determined and executed. The main components of SON Autonomic Layer are described as follows.

6.7.1 Monitor and Analyzer

This sublayer collects the information provided by sensors. Then, this information is aggregated and correlated in order to extract the relevant information. Then, the analyzer uses the relevant information to detect network situations (botnet detected, QoS/QoE degradation, DDoS attack, link failure). The whole process is organized in thee steps: Monitoring and Discovery, Aggregation/Correlation and Analyzer.

- **Monitoring and Discovery**. It collects the data sent by the SON Sensors. For this purpose, when a new Sensor is instantiated, it receives the notification and instantiation details and establishes a connection in order to receive the corresponding metrics. Moreover, it also receives the information provided by the physical and virtual sublayers. Then, the information is stored in a database in order to be processed by upper layers.

- **Aggregation and Correlation**. It performs the correlation and aggregation of the information stored in the monitoring and discovery database. This process involves additional actions, such as the data normalization, verification and removement of redundant information. At the end of this stage, only relevant information will be processed by the Analyzer module.

- **Analyzer**. Its main purpose is the comprehensive analysis of the relevant information provided by Aggregation and Correlation. The analysis also includes the prediction of future network problems. The network problems are known as HoN due the global vision or system-level scope of the analysis. For this purpose, it

takes advantage of several prediction, pattern recognition algorithms and big data techniques. The trending and predicted values for the metrics enable the application of proactive and reactive actions in the system. At the end of this stage, the network events are sent to the autonomic manager in order to establish the corresponding actions in the network.

6.7.2 VNF Onboarding

It acts as a repository of the different NFs. In this sublayer, the available NFs are stored and their capabilities are disseminated to the other sublayers. Similarly, the service providers can design, create and update their own applications. In this conext, the encapsulation of NFs follows the recommendations of the ETSI MANO framework for the NFV [ETS14]. Consequently, the NFV Manager (VNFM) is the key component for the lifecycle of SON sensors and actuators. The VNFM lifecycle exposes a common set of primitives for the automated instantiation, configuration, re-configuration and termination of the different VNFs. A common API enables service providers the easy design and development of their solutions. Once a solution NF is onboarded, the autonomic manager can use their capabilities to provide the new service (sensor/actuator).

6.7.3 Autonomic Manager

It uses different algorithms to diagnose the root cause of a network problem in terms of the HoN metrics provided by the Analyzer. Once the cause is detected, the autonomic manager uses the available NFs provided by the VNF Onboarding to decide the best reaction strategy or a countermeasure (e.g. deploy a new balancer, firewall or DPI). Then, the taken actions are sent to the NFV Orchestration and Management Layer. The related tasks are organized in three well defined modules.

- **Diagnoser**. It diagnoses the root cause of the network situations notified by the analyzer. For this purpose, it uses the information available on Monitor & Analyzer sublayer (topology, sensor data, HoN metrics) and takes advantage of stochastic algorithms, artificial intelligence, data mining to estimate the location of the source of the problem. Then, the root cause is notified to the Decision Maker.

- **Decision Maker**. It takes the incoming diagnosis information and decides a set of reactive and proactive actions to be taken into the network in order to avoid the detected and emerging network problems, respectively. Similarly, it also takes advantage of the integration of artificial intelligence algorithms to determine the responses or tactics to be taken. The taken decisions are notified to the action enforcer.

- **Action Enforcer**. It provides a consistent and coherent scheduled set of actions to be taken in the infrastructure. In other words, it validates, organizes and refines the tactics to avoid conflicts, duplications and nonsense order of actions. At the end of this stage, a high level description of the location, type of SELFNET SON Actuators, related configuration parameters are transferred to the orchestrator.

6.8 NFV Orchestration and Management Layer

This layer is responsible of the control and chaining of the different NFs in the virtualized infrastructure. The architecture follows the ETSI MANO [ETS14] recommendations and, consequently, it is composed of: Orchestration, VNF Management and VIM. As described in the Section 6.7.2. The VNF Management operations are partially developed in the VNF Onboarding. The other operations are described as follows:

- **NFV Management & Orchestration**. It is responsible of receiving the set of actions of the Autonomic Manager and orchestrate the NFs in the available virtual resources. The coordination and schedule of the enforcement of different actions is executed by the interaction with the virtual infrastructure manager.
- **Virtual Infrastructure Manager VIM**. It is responsible of organize and provide the virtual resources for the instantiation of the different NFs. The VIM interacts with the physical and virtual infrastructure to ensure the availability of resources and perform the automatic deployment of services.

6.9 SON Access Layer

This layer will provide an appealing and intuitive interface that provides different monitor and operation capabilities depending on the authorized users. In this way, the users can check on the current health status of the SELFNET operations. Similarly, the Access API will list the SON Sensor and Actuator currently deployed in SELFNET as well as the logging of a messages in order to enable a wider view of the SELFNET status. This interface is used by external actors such as BSS or OSS.

As described in the previous sections, SELFNET aims to be an independent and autonomous solution that acts mitigating or solving network problems without any actions from real users. In this way, the SON Access Layer also provides users with the study of the actions taken by SELFNET allowing the validation of corrective measures of the applications.

6.10 Summary

This chapter summarizes the self-management advances with SDN/NFV principles. Then, the Self-Organized Network Management in Virtualized and Software Defined Networks Project SELFNET is proposed. Next, the different SELFNET layers and sublayers are described. The infrastructure layer, data network layer, SON control layer, SON autonomic layer, NFV orchestrator & management layer and SON access layers are discussed.

Part II

Description of the Research

Esta parte del documento corresponde a la
aportación original y exclusiva de la Tesis Doctoral.
This part of the document corresponds to the
original and exclusive contribution of the Doctoral Thesis.

Chapter 7

Optimized Monitoring and QoS in SDN Networks

The centralized control of the network and the separation of data and control planes proposed by SDN have changed the rigid, static, and complex nature of the networks. The decisions taken by the control plane depends on the accuracy of the monitored information on network performance and detection of network events (link failure, delay, loss, network overhead). However, the monitoring information is typically provided by external network monitoring solutions which require the installation of specialized (and costly) equipment.

Firstly, this chapter presents an efficient SDN monitoring framework using the OpenFlow protocol developed during the research. This framework uses profiling to provide different monitoring levels based on the requirements of the "network programmer". Moreover, the pluggable architecture enables the creation, updating and customization of high level metrics as well as the orchestrator balancing the monitoring tasks and offering an adaptive method of polling information requests based on the load of the controller and the size of the network. The implementation of network performance metrics (data rate, loss rate and delay) and the results of experiments using video streaming traffic demonstrate the effectiveness of the framework.

The increase of data traffic moving in the Internet is particularly concentrated in multimedia content [zet16]. However, the rigidity of the typical Internet architecture based on best-effort oriented IP network has limited the development of innovative network services. The new post-PC multimedia services require networks with high levels of flexibility and customization in order to provide efficient security, mobility, availability and QoS. In this context, SDN is a novel paradigm that decouples the data and control plane in network devices and opens the programming capabilities of the network behavior. SDN opens the possibility to customize the network behavior in function of the different user requirements.

Secondly, a SDN framework to provide QoS for different multimedia services is presented. This framework uses OpenFlow, Network Virtualization and establishes functional boxes and interfaces to test different routing algorithms. Then, the advantages of the framework are tested with the implementation of a network performance and routing algorithms functions. The experiments with video streaming information show a quality

optimization (PSNR, SSIM, *Mean Opinion Score* (MOS)) in comparison with the best effort engine. Finally, the challenges of SDN and the future lines of work are presented.

The rest of this chapter is organized as follows: Section 7.1 describes the framework for optimized monitoring. Similarly, the framework for optimized QoS routing is explained the Section 7.2 Section 7.3 presents the implementation of the optimized monitoring and QoS frameworks. Finally, Section 7.4 summarized this chapter.

7.1 Framework for Optimized Monitoring

The use of a monitoring framework within the own SDN architecture has several advantages. Firstly, the implementation is independent of the hardware or specific vendor. Once the equipment is compatible with a specific southbound API (e.g. OpenFlow), the complete network can be monitored by the controller.

This affirmation does not exclude the use of other network monitoring solutions. Additionally, the rapid response of the controller to network events is fundamental to guarantee an efficient network performance. With more proximity between monitoring and the controller, more accurate and timely decisions can be made. However, the use of the controller to execute monitoring tasks presents several risks that must be considered. For instance, it is necessary to minimize the CPU load in order to avoid interferences in other processes of the controller. In other words, the constant monitoring of all variables in all switches is highly inefficient. It also generates an exponential increment of message requests increasing the load in the data plane (switch hardware). Furthermore, the information required by the controller depends on the network application. For example, real time streaming applications are highly sensitive to throughput and delay variations, while for e-mail services, these measures are basically irrelevant. Moreover, it is also desirable that the algorithms used to monitor these values can be easily added, modified or updated as well as a clear presentation of the results.

Taking into account these aspects, we propose an optimized monitoring framework illustrated in Figure 7.1. The proposed framework consists of different functional boxes within the control and application layer of the SDN architecture. The infrastructure layer includes the switches, routers and other elements responsible for the forwarding tasks. The control layer makes decisions about the network behavior and sends these instructions to the infrastructure layer through a southbound API (OpenFlow). The control layer offers functionalities to the application layer thanks to a northbound API (Rest API), where high level policies and applications are implemented.

The functional components of the framework are explained in detail below:

- **Users requirement** The application that needs information about the network creates a new monitor profile. In this profile, the user specifies the user ID, type of metrics and the level of accuracy. This information is sent to the control layer through a northbound API (RestAPI). This ensures that the application will receive updated information of the network performance based on the required metrics. The user can change the configuration of the profile without affecting other modules or changing the network behavior.

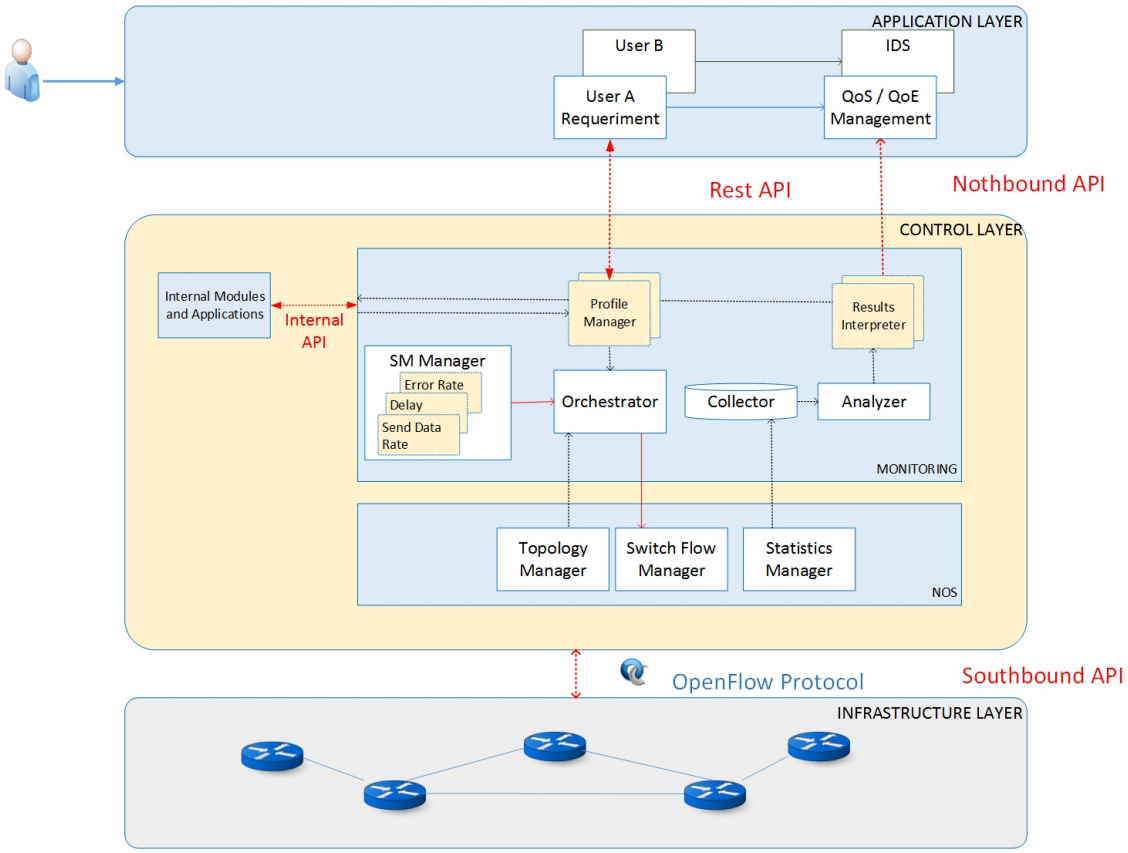


Figure 7.1: Monitoring framework

- **Profile Manager**. Receives the different requirements of the application layer and creates a monitoring profile for each user. Additionally, this manager also registers the requests of modules located within the control plane (internal modules) through an internal API. The Profile Manager also updates the Results Interpreter module in order to synchronize and provide accuracy in the results sent to the users.

- **Topology Manager**. This module uses the LLDP (Link Layer Discovery Protocol) and analyzes the OpenFlow messages to identify the network devices, their capabilities and the links presented in the infrastructure layer. The topology is organized as a graph $G(N, A)$, where N represent the switches and A the links present in the infrastructure. The information of the topology is sent to other modules of the control layer.

- **SM Manager**. The Store Metrics Manager consists of a container that registers and stores the available monitoring algorithms. These algorithms are pluggable and can be easily added, modified and updated. The SM Manager can include own or third parties modules. The modules use OpenFlow as southbound and can include external APIs to receive information of other external monitoring tools (NetFlow, SFlow).

- **Orchestrator**. This module organizes and schedules the different monitoring

modules in function of the requirement of the profile manager. For this purpose, it takes into account the information provided by the topology manager and balances the load of requests in the infrastructure layer. This module also attempts to reduce duplication of request and avoids data redundancy.

- **Switch Flow Manager**. Receives the instructions of the monitoring algorithms organized by the Orchestrator and sends the OpenFlow messages to request information to the switches. Furthermore, this module uses the topology manager information and modifies the flow tables in case of establishing a route for probe packets.
- **Statistics Manager**. Receives the information provided by the OpenFlow messages and recovers the statistical information of these messages. This information is sent to the Collector module.
- **Collector**. This module filters and organizes the information provided by the statistics manager and stores the relevant information. This module can use compression, data mining and search algorithms to reduce the space required to store the data.
- **Analyzer**. Reads the results from the Collector and carries out event processing, data correlation, learning algorithms to provide high level metrics. The high level metrics are focused not only to individual devices (switch) but also in a global view or complex distributed grouping of devices (network domain). Similarly, the analyzer can infer trending and predicted values about the future network behavior.
- **Results Interpreter**. This module receives the results of the analyzer module and infers HoN events such as actual or future ACL-violations, IDS/Firewall alerts, DDoS attack, intrusion detection or anomalous behavior. Also, it decides if there has been a traffic increase/decrease, and can ask for a bigger bandwidth if necessary.
- **Applications**. The different network applications receive only the relevant high level monitoring information. The users can change the monitor profile in accordance with their needs and dynamically modify the requests on switches. Furthermore, the user is able to apply reactive and proactive actions providing dynamic responses of different network events.

7.2 Framework for Optimized QoS Routing

The proposed framework consists of different functional boxes as described in Figure 7.2. The global architecture includes the infrastructure, virtualization, control and application layers. The infrastructure layer represents the elements responsible for the forwarding process (switches, routers and other data plane elements). The virtualization layer [CB10] logically abstracts the data plane resources of the infrastructure layer. Then, the service provider establishes different "slices" of logical topologies and assigns each slice to the different users. FlowVisor [SGY+09] is an example of network virtualization in SDN

networks. Similar to virtualization in computer systems, this virtualization layer is transparent for the upper layers. For this reason, the control layer can use OpenFlow protocol as Southbound API. The functional boxes of the control layer are based in the "per flow" separation of traffic and the information provided by the OpenFlow Protocol. Additionally, the framework uses Rest as Northbound API to communicate with the application layer. The network programmer creates network applications based on the functions provided by the control plane.

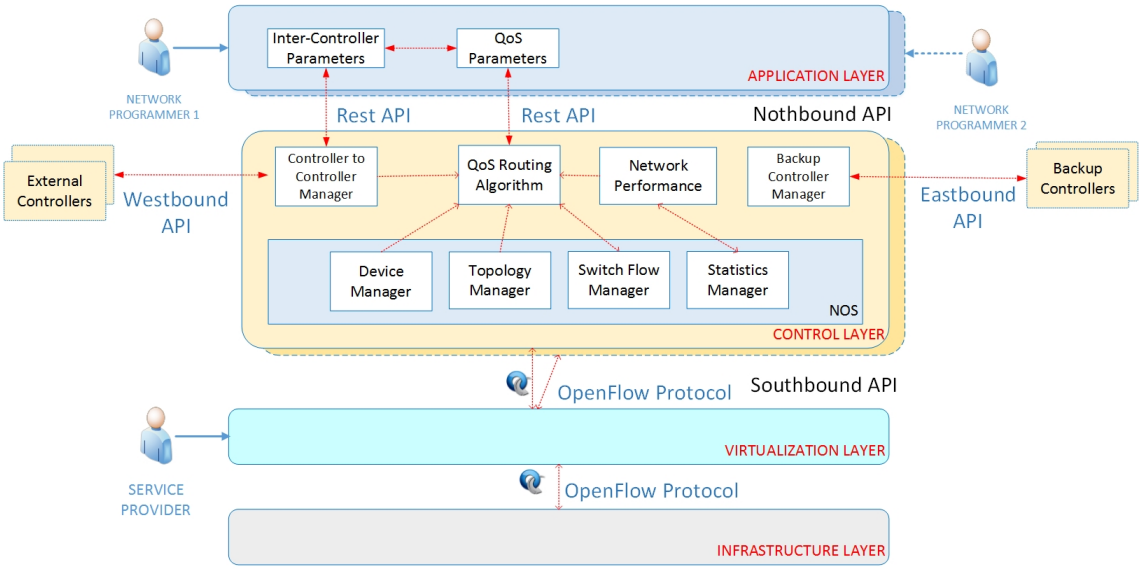


Figure 7.2: Proposed QoS framework

The descriptions of the application layer functional boxes are defined as follows:

- **QoS Parameters**. This module configures the different parameters to the QoS routing module. These parameters can include data types, priorities, QoS levels, devices, amongst others. These values are sent to the QoS Routing Algorithm through a well-known northbound API (RestAPI).
- **Inter-Controller Parameters**. The user establishes the parameters and the information to be sent to another network or external SDN controller.

The descriptions of the control layer functional boxes are defined as follows:

- **Topology Manager**. Recognize the network devices (switches, links) present in the infrastructure layer and their capabilities. Topology Manager uses the information of OpenFlow messages and LLDP (Link Layer Discovery Protocol) to infer the topology. The Topology is represented as a graph $G(N, A)$ where N is the set of switches and A the set of arcs (links). This information is available to other functions and modules of the controller.
- **Device Manager**. Identifies and tracks physical hosts or terminal devices (server, client, mobile phone, webcam, sensor) and their location in the network. The

location of a device within the network is saved as the switch id and port id where it is connected $(Ni, pi) \in G$. Device Manager Service can find the position of a device based on his MAC or IP address and send this information to other modules.

- **Controller to Controller Manager**. Establishes the communication with other controllers through a westbound API. This API can be used to send information to other networks (e.g. Autonomous System AS). This module can learn the existence of new hosts or final devices connected externally. This information is sent to the QoS routing algorithm.
- **Backup Controller Manager**. Periodically send updates to backup controllers. In case of Denial of Service attack or a NOS failure, the backup controller can automatically take control of the whole network.
- **Statistics Manager**. Read and organize the statistical information sent by the counters of the switches. This information can be analyzed to detect failures or congestion of links.
- **Switch Flow Manager**. Receive the paths and priorities calculated by the QoS Routing Algorithm and configure the flow tables of the switches.
- **Network Performance**. Uses the information provided by the statistics manager and executes an algorithm to evaluate the performance of the network. Different performance algorithms can be implemented in order to improve the accuracy of the results. The variables can include bandwidth, data rate (bytes or packets per second), packet loss rate, delay, among others. This information is continuously being sent to the QoS Routing Algorithm for the path calculation.
- **QoS Routing Algorithm**. Read the QoS parameters provided by the network administrator and calculates the appropriate paths for the information circulating through the network. This algorithm reads the topology, the devices connected and also the actual performance of the links. This means that the controller can also automatically react in case of a failure of degradation in the quality of the network. Similarly to network performance engine, this algorithm can be improved due to the other functional boxes working independently.

7.3 Implementation

In this section, we explain the process used to implement the monitoring and QoS frameworks.

7.3.1 Network Operating System NOS

The function of the NOS is similar to the concept of Operating System O.S. in computing. That is, the NOS provides a high-level abstraction of information, resources and hardware. It facilitates the creation of applications independently of the hardware design using a

high level programming language. Moreover, the NOS receives and sends the OpenFlow messages and provides basic functions to the other modules. Examples of available NOS include NOX [GKP+08b], Maestro [NCC10], Floodlight [flo17], among others.

The framework is implemented using the Floodlight controller. Floodlight is based on Java, available with Apache License and extensively supported by a developer's network. It enables a modular programming together with other advantages of Java environment (platform-independent, timers, threads, libraries). Moreover, Floodlight implements some basic modules compatible with OpenFlow protocol version 1.0 such as topology manager, device manager or statistics manager. These modules are available through a Java API to other internal control plane modules as well as a Rest API to remote application modules.

7.3.2 Topology Abstraction

The switches and links present in the network infrastructure is represented as a graph $G(N, A)$, where N represents the connected devices and A the arcs between them (links) [AMO93]. In this context, the term $arc(i, j) \in A$ symbolizes a link between nodes i and j. Each link present in the topology can be associated with a weight value or link cost. For instance, the term c_{ij} represents the cost associated with the link $arc(i, j)$. The link cost can represent a fixed value (hop count ($c_{ij} = 1$)) or a time-varying field, such as jitter, loss or available bandwidth. R_{sd} represents the set of paths or routes available from the source s to destination d. It is also assumed that the network is directed, the network contains a directed path from node s to every other node in the network and the network does not contain a negative cycle.

7.3.3 Monitoring Framework

7.3.3.1 Orchestrator

These elements are responsible for organizing the algorithms used to establish the network behavior. The algorithms are ordered in modules in such a way that they can be easily added, modified or removed without interfering with each other. The SM Manager registers and organizes the modules used to monitor the network, while the Orchestrator authorizes the execution of the SM modules and controls the load of requests on switches. The optimization of the Orchestrator as well as the algorithms of the SM Manager are open challenges and can be constantly improved. Moreover, it is clear that the monitoring tasks can affect the whole performance of the network, so it is required that the orchestrator can control the load of monitoring tasks. In this implementation, the orchestrator will control the load regulating the pulling period of the requests in function of the network size and the controller capacity. The logic of this technique is explained in the Algorithm 1. The user will assign a minimal period monitoring time t_{min} and the load capacity of the controller α. In case of $\alpha = 1$ the controller is able to support high level of load and with α close to 0 the controller capacity is highly limited. For its part, the network size is estimated in function of the number of devices (N) and links (A). All these variables are taken into account to balance the pulling period (t_{orc}).

Algorithm 1: Orchestrator function

Input: network graph $G(N, A)$
minimal period monitoring time t_{min}
controller capacity $0 < \alpha \leq 1$
Result: optimized monitoring time t_{orc}

```
1 procedure OrchFunction
2     calculate the optimized time in function of the network;
3     if α = 1 then
4         t_orc=t_min;
      else if 0 < α ≤ 1 then
5         if A ≤ N then
6             t_max=N*t_min;
          else
7             t_max = (N + A/N).t_min
8         t_orc = (1 − α).t_max
9 end procedure
```

7.3.3.2 SM Manager

The SM Manager can include own or third parties modules. As detailed in the Related Works section, there are notable OpenFlow -based monitoring algorithms [CBAB14], [vADK14], [KGH13] that have been taken into account as a baseline. The present work presents 3 Algorithms allowing us to obtain the metrics of send data rate (dr_{ij}), packet loss rate (lr_{ij}) and delay (d_{ij}). These proposals have been integrated as modules of SM as follows.

The send data rate is explained in the *SendDataRate* procedure in Algorithm 2. *SendDataRate* uses the passive method and sends port request to switches and reads the responses from them. The switch request is sent through the controller to switch message OFPT_STATS_REQUEST. The controller sends this message in an optimized period monitoring time of t_{orc}. The switch responds with an OFPT_STATS_REPLY message. The controller uses the function ofp_port_stats to receive the response, identify the counters with the corresponding switch (src node) and port (src port) in the topology and saves this information s_i^k. Then, the send data rate $dr_{i,j}$ for src node - src port is the difference of the sent bytes counter $(s_i^k - s_i^{k-1})$ in the time period t_{orc}.

Similarly, the packet loss rate metric is explained in the *PacketLossRate* procedure (Algorithm 3). *PacketLossRate* is a passive method that also takes the information from the OFPT_STATS_REQUEST message. That is, the controller does not need to send additional requests due to it using the same information saved to calculate the send data rate. For its part, the implementation uses the Topology Manager to discover the corresponding links present in the infrastructure. Each link $arc(i, j)$ is represented with the key (source node - source port - destination node - destination port). With this information, the *PacketLossRate* calculates the difference lr_{ij} of sent bytes in source node - source port s_i^k with the corresponding received bytes from the destination node - destination port r_j^k in the monitoring time period t_{orc}.

Algorithm 2: Send data rate (SM manager)

Input: network graph $G(N, A)$
optimized monitoring time t_{orc}
Result: send data rate dr_{ij} for each $arc(i, j) \in A$

1 **procedure** SENDDATARATE
2 Start timer k with period t_{orc};
3 **foreach** *period k = 0, 1, 2, 3, ...* $\in t_{orc}$ **do**
4 **foreach** $arc(i, j)$ **do**
5 Read the sent bytes s_i of the source link port with s_i^k=*tx_bytes* in *ofp_port_stats(src node,src port(i))*;
6 **if** $k > 0$ **then**
7 Calculate the data rate of the link $dr_{ij}=\frac{s_i^k - s_i^{k-1}}{t_{orc}}$;
8 **end procedure**

Algorithm 3: Packet loss rate (SM manager)

Input: network graph $G(N, A)$
optimized monitoring time t_{orc}
Result: packet loss rate lr_{ij} for each $arc(i, j) \in A$

1 **procedure** PACKETLOSSRATE
2 Start timer k with period t_{orc};
3 **foreach** *period k = 0, 1, 2, 3, ...* $\in t_{orc}$ **do**
4 **foreach** $arc(i, j)$ **do**
5 Read the sent bytes s_i of the source link port with s_i^k=*tx_bytes* in *ofp_port_stats(src node,src port(i))*;
6 Read the received bytes r_j of the destination link port with r_j^k=*rx_bytes* in *ofp_port_stats(dst node,dst port(j))*;
7 **if** $k > 0$ **then**
8 Calculate the packet loss rate of the link with $lr_{ij}=\frac{(s_i^k - s_i^{k-1}) - (r_j^k - r_j^{k-1})}{t_{orc}}$;
9 **end procedure**

In addition, the *Delay* module uses an active method to calculate the time delay. This procedure is explained in the Algorithm 4. At first, the controller takes a timestamp value t_{st}, encapsulates this value in the load of a packet probe Pp, and then sends this information through the link in the source switch i. The controller sends instructions to the switches to identify this probe packet and sends it back to the controller. In this implementation, we use an experimental network protocol number (253) as the key to identify Pp packets. When the packet arrives in the other extreme of the link (destination node j), the switch identifies, encapsulates and sends this packet back to the controller pkt_{in}. The controller identifies the packet, recovers the initial timestamp t_{st} and then compares this value with the actual timestamp t_{ctr}. The delay value is estimated as the difference between the values t_{ctr} and t_{st} for each link $arc(i, j)$.

Algorithm 4: Delay (SM manager)

Input: network graph $G(N, A)$
optimized monitoring time t_{orc}
Result: delay d_{ij} for each $arc(i, j) \in A$

```
1  procedure DELAY
2      procedure SENTPROBEPACKET
3          Start timer k with period t;
4          foreach period k = 0, 1, 2, 3, ... ∈ t do
5              foreach arc(i, j) do
6                  Read the actual timerstamp t_st of the controller
7                  t_st = Date.getTime()
8                  Encapsulate t_st within a Probe packet Pp
9                  Pp_{i→j} = new packetOutMessage()
10                 Pp_{i→j}.setProtocol(253)
11                 Pp_{i→j}.setData(t_st)
12                 Sent the Probe packet to the source switch
13                 switch(i).write(Pp_{i→j})
14     end procedure
15     procedure RECEIVEPROBEPACKET
16         Verify that the incoming packet message is part of a probe packet
17             if packetInMessage pkt_in is a Probe packet (Pp_{i→j}) then
18             Read source, destination and timestamp
19             t_st = pkt_in.getData()
20             i = pkt_in.getSource()
21             j = pkt_in.getDestination()
22             Read the actual timestamp of the controller
23             t_ctr = Date.getTime()
24              Calculate the delay value of the link arc(i, j)
25             d_ij = t_ctr − t_st
26     end procedure
27 end procedure
```

7.3.4 QoS Framework

7.3.4.1 Network Performance

The active measurement of the performance of the network is an open challenge. The statistics manager module provided by the NOS can read the theoretical maximal data rate of a particular port through the OFP_PORT_FEATURES parameter. However, the real capacity or maximal data rate depends on the conditions and quality of the link between switches. Furthermore, the OpenFlow 1.0 specifies counters in the switch at different levels (table, port, flow, and queue) but does not provide packet delays or jitter measurements. Moreover, the controller reads this information with the inevitable time delay of the link controller-switch. The models proposed in [KGH13] [STH14] presents algorithms to improve the performance diagnosis for OpenFlow networks. Another solution is the

integration of data plane measurements or specialized tools such as sFlow [GAA+13].

In our model, the cost of the Network Performance module uses the engine of the Algorithm 5. First, the module initializes the cost function with the value of 1, assuming that the links do not have a loss rate. Then, the module initiates a timer to continuously monitor the port statistics counters (OFP_PORT_STATS) and periodically collects the sent and received bytes in the ports of the switch. Using the information of the topology (network graph) and the information of the counters, the algorithm establishes the data rate and packet loss rate. The network performance function considers the ideal condition $s^k > s^{k-1}$ and $r^k > r^{k-1}$. The data and packet loss rate algorithms are previously presented in the previous sections. The cost function uses this information and an adjustment factor $\propto$ to calculate the dynamic cost of the link. This value is continuously updated and sent to the QoS Routing Algorithm.

Algorithm 5: Network performance function

Input: network graph $G(N, A)$
period monitoring time t
link bandwidth bw_{ij} for each $arc(i, j) \in A$
data rate dr_{ij} for each $arc(i, j) \in A$
packet loss rate pl_{ij} for each $arc(i, j) \in A$
adjustment factor $\propto$

Result: cost c_{ij} for each $arc(i, j) \in A$

```
1 procedure CostFunction(I)
2     Start timer k with period t;
3     foreach period k = 0, 1, 2, 3, ... ∈ t do
4         foreach arc(i, j) do
5             if k = 0 (initial time) then
6                 c_ij=1;
7             if k > 0 (each period) then
8                 Calculate the cost of the link with
                  c_ij=1+(∝.dr_ij+(1−∝).pl_ij)/bw_ij ;
9 end procedure
```

7.3.4.2 QoS Routing

There are multiple techniques to provide QoS in typical hop-by-hop network architectures [VMK04]. However, the global vision of the network provided by SDN and the "per-flow" OpenFlow engine can facilitate new models of efficient Routing Algorithms. In this work, we present a basic solution for QoS Routing based on graph of nodes and links $G(N, A)$. This procedure is summarized by Algorithm 6. In this implementation, the QoS Routing module can assign different packet fields as QoS parameters (w/QoS), for example TCP or UDP port, IP source, IP destination, MAC source, MAC destination, among others. The user provides this information to the Routing module through the RestAPI. For example, the user can establish high priority (w/QoS) to video streaming that use a specific port number (e.g. UDP Port = 5532) and best effort engine and normal priority (w/oQoS) to

other users.

The module also receives the link cost values c_{ij} of the network performance module and the graph $G(N, A)$ of the topology module. When the controller receives a PACKET IN p message, it analyzes the packet header h_p and establishes if the packet fulfills the QoS requirements w/QoS. In the example, the controller verifies if the UDP port number is 5532. If so, the module reads the switch s_s from where p comes and the IP destination d_{IP} in order to locate the position of destination s_d using Device Manager module. Then, this implementation uses the Dijisktra algorithm [VMK04] to establish the route r_{sd} of the network in function with the link costs provided c_{ij} by the Network Performance module based on data rate and loss rate of the links. In the present implementation, the delay is not relevant for the considered sensing application domain. The addition of new metrics is part of the future work. The chosen route will be the path with the dynamic minimum cost.

This path is then installed in the network devices through the Switch Flow manager. The Switch Flow manager creates F OpenFlow flowmod messages to configure the flow tables of the switches and assign (w/QoS) packets with higher priority. In case of (w/oQoS), the module uses the Floodlight IRouting service to establish a regular path between source and destination (minimum hop count).

Algorithm 6: QoS routing function

Input: network graph $G(N, A)$
link cost c_{ij} for each $arc(i, j) \in A$
parameters to identify QoS packets w/QoS
(e.g. UDP port, IP src, IP dst, MAC src, MAC dst)
received OpenFlow PACKET IN messages p

Result: Set of OF-flowmod messages F to configure switches

1 **procedure** QoS Route
2 Read the header of the PACKET IN message p with
h_p=*read_header*(p) ;
3 Check if h_p is part of w/QoS (e.g. UDP Port = 5532) with
if $h_p \in w/QoS$ **then**
4 Read the OF-Switch s_s from where p comes with
s_s=*read_sw_source*(p) ;
5 Read the destination IP address d_{IP} from h_p with
d_{IP}=*read_dst_ip*(h_p) ;
6 Use Device Manager Service to find OF-Switch s_d
where d_{IP} is connected with
s_d=*DM_find_device_ip*(d_{IP}) ;
7 Find route r between source s_s and destination s_d
using links cost c_{ij} with
r_{sd}=*Dijkstra*(G,s_s,s_d,c_{ij}) ;
8 **foreach** *OF-Switch* $s_k \in r_{sd}$ **do**
9 Create OF-flowmod F_{sk} for OF-Switch s_k
with high priority with
F_{sk}=*create_flowmod_message*(s_k) ;
F_{sk}.*set_high_priority*(); ;
else
10 Process packet p with best effort
engine and low priority with
F_{be}=*fdl_irouting_service*(p) ;
F_{be}.*set_normal_priority*() ;
11 **end procedure**

7.4 Summary

This chapter presents the framework for optimized monitoring for SDN networks. Similarly, the framework for optimized QoS routing is described. The implementation of the proposals in terms of NOS, topology abstraction and the tasks performed by the functional boxes of the frameworks are explained.

Chapter 8

Results

This chapter presents the results of the experiments of the Optimized Monitoring and QoS frameworks. This chapter is organized into 3 Sections. Section 8.1 describes the application scenario and results of the optimized monitoring framework. Section 8.2 gives details on the application scenario and results of the optimized QoS framework. Lastly, Section 8.3 summarizes this chapter

8.1 Optimized Monitoring Framework

8.1.1 Application Scenario

The feasibility of the framework is tested using the topology described in Figure 8.1. This topology is emulated using python scripts in the Mininet emulator tool 2.1.0 [LHM10]. Mininet enables the creation of custom topologies of OF-switches and links within a simple laptop (Intel core i5 2.4 Ghz, 8GB DDR3 RAM, OS X 10.10). The tests are executed in a VM Linux Ubuntu 13.04 (64 bits, 4 Gb RAM, 2 CPUs, 8 GB HD). The topology is composed of 3 switches and 3 host connected in a linear topology. The links L1 (s1-s2) and L2 (s2-s3) are configured with values of maximal data rate, loss percentage and delay.

The host h1 uses VLC video server to send a video file "highway_cif" [vid17] (MPEG $\frac{1}{2}$, 2000 frames, 80 s, 2.97 MB) to the client h3 using RTP/UDP streaming protocol in order to model traffic. The Floodlight controller executes a learning switch module to establish the path between source and client. Furthermore, the monitoring module is configured with a monitoring time of 200 ms. and a controller capacity $\alpha = 1$. The experiments are executed in the worst-case measurement scenario, that is, the error rate and delay are introduced simultaneously. The representation of the monitoring information provided by the RestAPI northbound is described in Figure 8.2. The RestAPI interface provided by the framework enables the rapid development of high level network applications.

8.1.2 Results

The test is repeated 20 times and the average values of data rate, loss rate and delay estimated by controller are evaluated. The results of the experiments are depicted in Figure 8.3, 8.4 and 8.5. In order to reduce distortions, we display the trend line with an

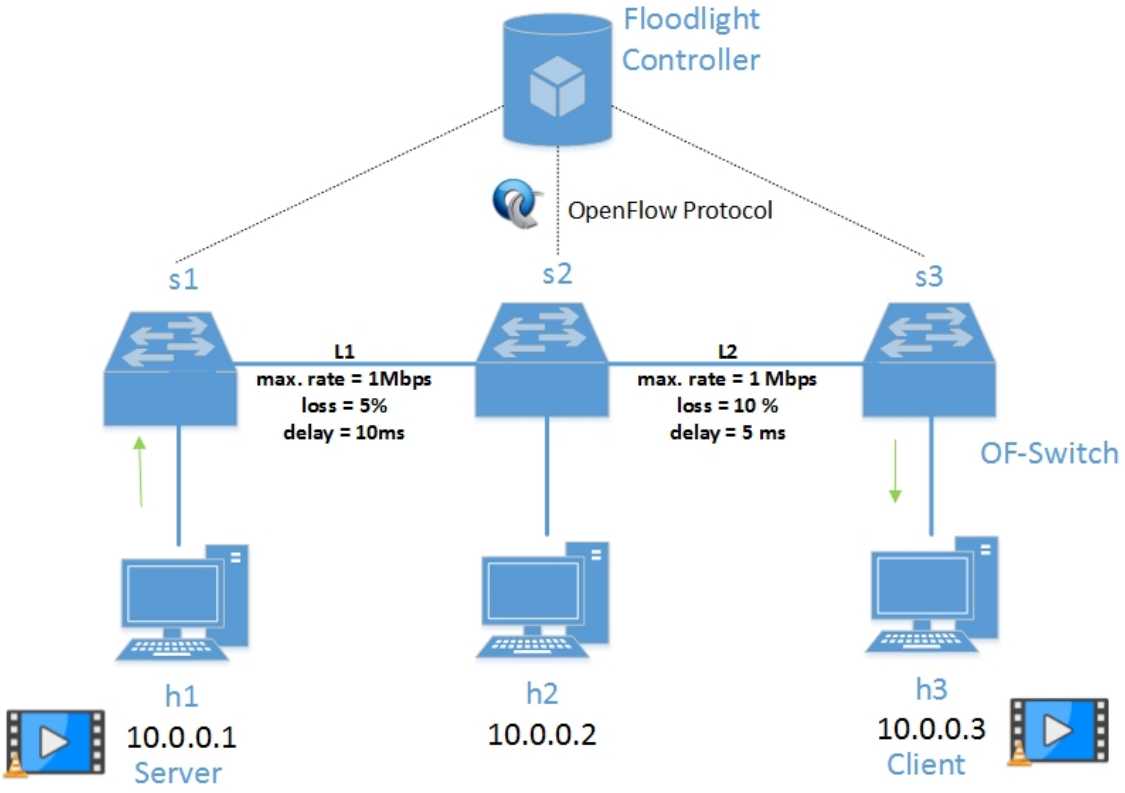


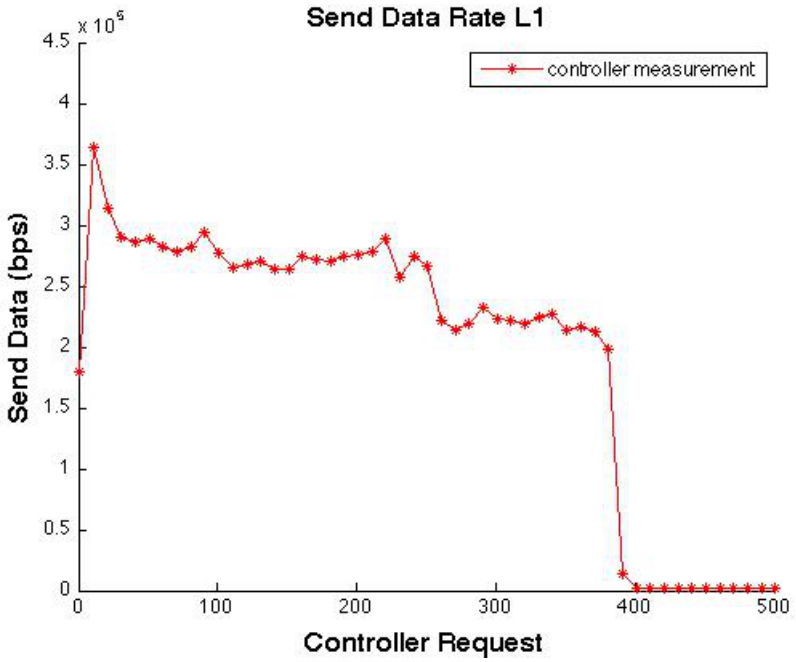
Figure 8.1: Optimized monitoring test topology

{"test": {"1":[{"src-switch":2 , "src-port":2 , "dst-switch":1 , "dst-port":2 , "lk-dataRate":1680 , "lk-errorRate":0 , "lk-delay":14} , {"src-switch":1 , "src-port":2 , "dst-switch":2 , "dst-port":2 , "lk-dataRate":275680 , "lk-errorRate":54800 , "lk-delay":14}], "2":[{"src-switch":3 , "src-port":2 , "dst-switch":2 , "dst-port":3 , "lk-dataRate":1680 , "lk-errorRate":0 , "lk-delay":15} , {"src-switch":2 , "src-port":3 , "dst-switch":3 , "dst-port":2 , "lk-dataRate":220880 , "lk-errorRate":44760 , "lk-delay":16}],"3":[]}}

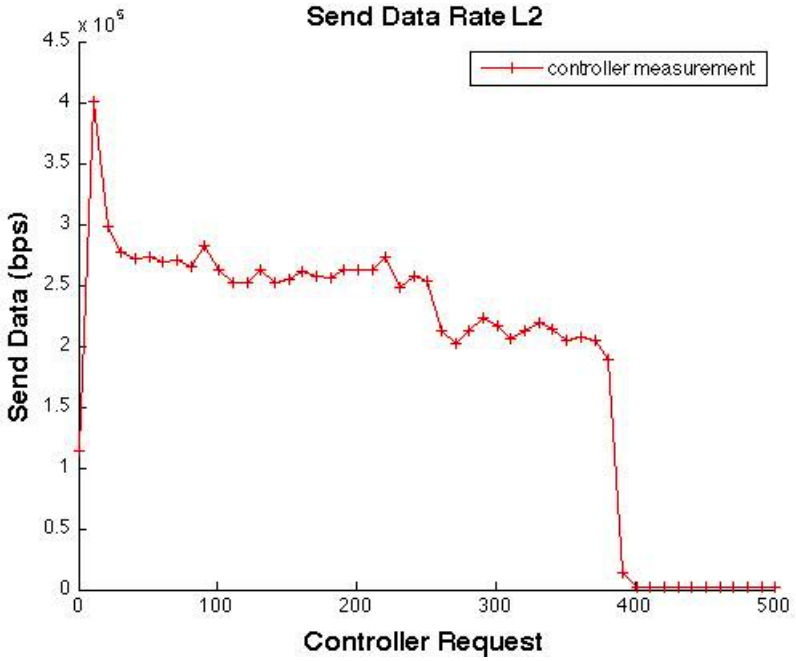
Figure 8.2: Interfaz rest API

average of 10 data. The Figure 8.3(a) describes the send data rate metric (calculated by the *SendDataRate* procedure) in bps of L1 (s1-s2) and the Figure 8.3(b) the corresponding metric of L2 (s2-s3). As expected, the measured send data rate detects an increase of traffic in the links caused by the video transmission between server and client. Once the transmission is finished (about 400 controller requests), the controller measures a minimal send data rate in both links.

Furthermore, the loss rate of the links L1 and L2 are depicted in Figure 8.4. The red line (solid) represents the loss rate percentage measured by the controller, while the blue line (dotted line) shows the *netem* loss percentage configured in mininet or data plane (L1=5%, L2=10%). In this case, the controller is also capable of detecting the loss of information between links. However, the transmission of video streaming information increases the variation between the data plane and the value estimated by the controller.



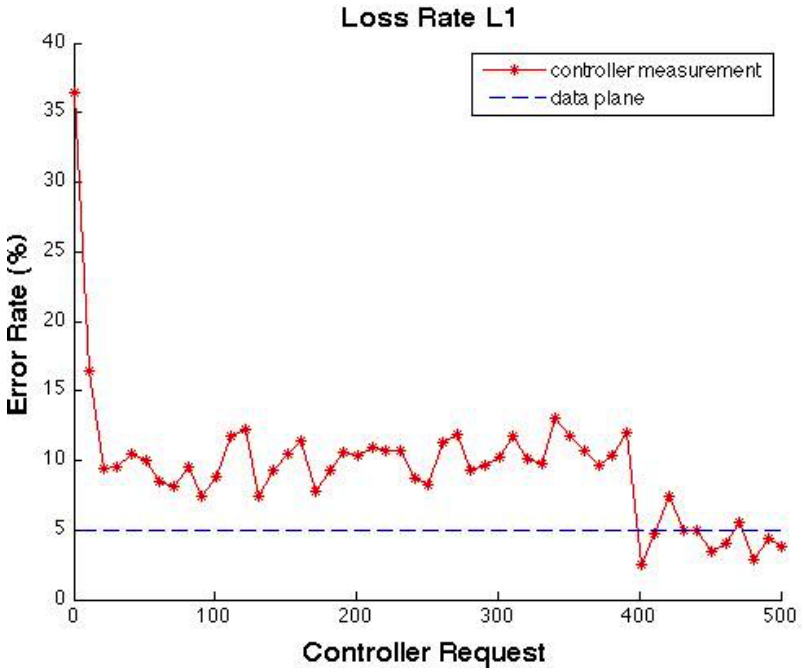
(a) Send data rate for L1 (s1-s2)



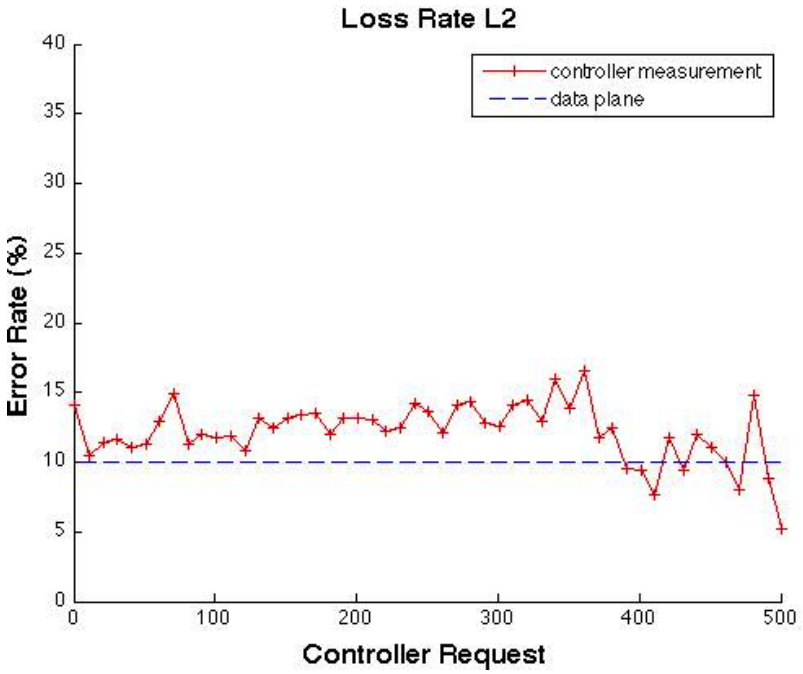
(b) Send data rate for L2 (s2-s3)

Figure 8.3: Send data rate measurement for L1 and L2

The controller delay estimation for L1 and L2 is shown in Figure 8.5. The blue line (dotted line) shows the data plane delay of L1 (10 ms.) and L2 (5 ms.) and the red line (solid) displays the value measured by controller. In both cases, it can be seen that the measured delay is slightly higher compared with the data plane. This effect is caused by the extra latency between the switch and controller. In other words, for instance,



(a) Loss data rate for L1 (s1-s2)



(b) Loss data rate for L2 (s2-s3)

Figure 8.4: Loss data rate measurement for L1 and L2

the probe packets in L1 is moving from controller to switch s1, then from s1 to s2 and the switch s2 send the packet back to the controller. This variation can be minimized calculating the delay present between controller and switch using the same active procedure (send and receive a probe packet from the same node) [vADK14]. However, this proposal

will increase not only the traffic in switch-controller data path, but also the switch and controller processing tasks.

Table 8.1 presents the nominal data plane of loss rate and delay measurements together with the mean and standard deviation of the results of the experiments. It is clear that, although those values present variations between data and control measurements, the controller can detect problems and quickly send alerts to applications in order to identify causes and take decisions to reduce negative impacts in the network behavior.

The performance of the framework is also verified in order to analyze the capacity of the controller in different size topologies. The controller capacity is directly linked to the equipment, programming language (Python, Java, C), work load, topology and forwarding complexity [TGG+12]. In the work presented in [VCBLGV13], we analyze the performance of different NOS.

Table 8.1: Summary evaluation of the experiment

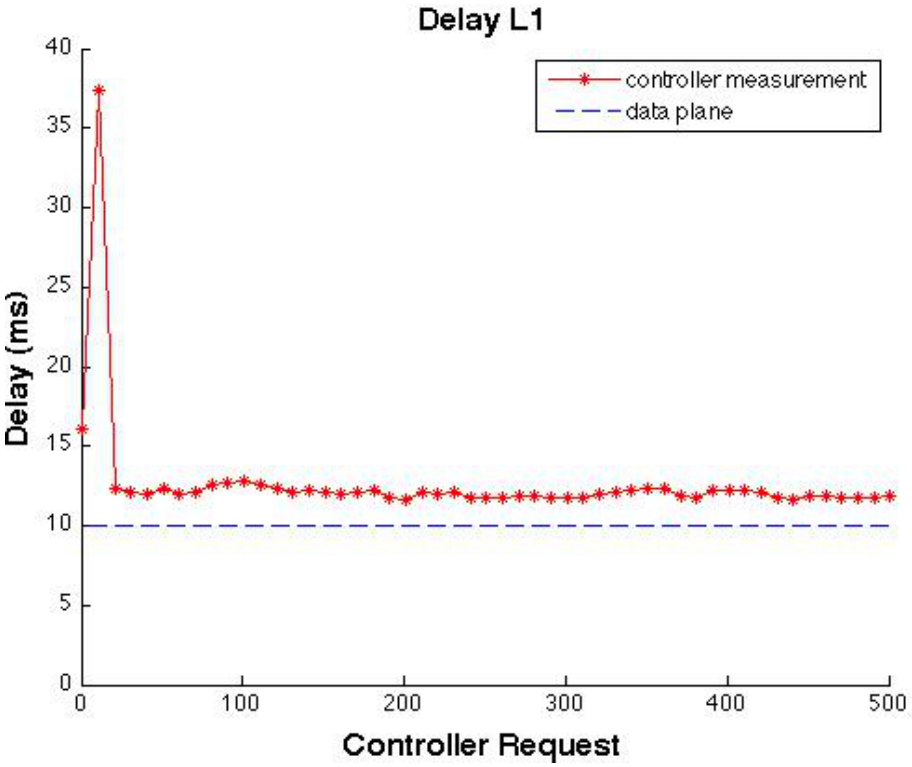
Link	**Loss Rate (%)**			**Delay (ms)**		
	data plane	mean	std. dev.	data plane	mean	std. dev.
L1 (s1-s2)	5	9.323	5.716	10	12.626	5.474
L2 (s2-s3)	10	12.623	6.158	5	8.147	0.674

In the present experiment scenario, Table 8.2 and Figure 8.6 show the CPU and Memory usage of the controller in different topology sizes, from 10 to 100 switches in a linear topology with an increase of 20 switches between each simulation. These values are measured through system monitoring request using the linux top service. The top requests are sent during the video streaming simulation between the hosts connected in the first and last switch respectively. The CPU usage shows a peak at the beginning of the simulation as a result of the controller setting up as well as the mutual interference between the processes (Mininet, Floodlight, VLC server and client) running in the same VM. Similarly, as expected, the increase of the network size has an incremental impact in the CPU usage and duration of the streaming simulation. The maximum value measured of CPU usage is 44.4 % for a topology of 100 sw. For its part, the memory usage receive a slight increase with higher topologies (1 %). However, these memory usage percentages are stable during the session. Therefore, one may conclude that the framework is suitable for small-medium scale topologies and in case of large topologies, the framework can be optimized with multi-controller platforms.

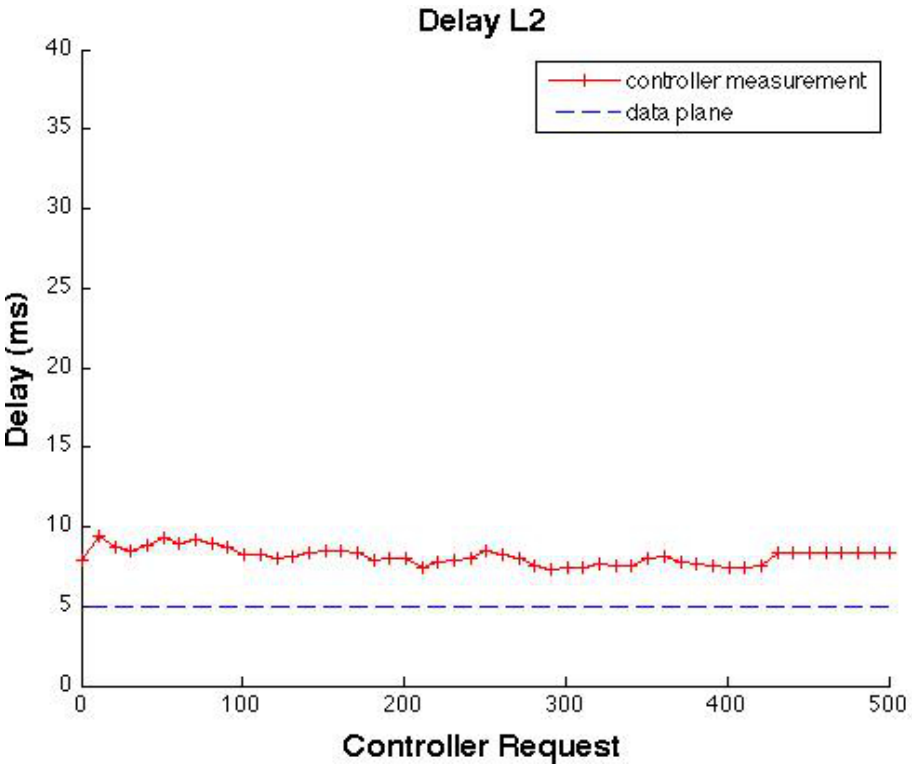
8.2 Optimized QoS Framework

8.2.1 Application Scenario

The framework is tested using the topology shown in Figure 8.7. Also, the test method described in [JSMR01] [EDBT12] has been taken into account as baseline, adapted and expanded to include new issues and metrics, as detailed below. The topology is emulated using a server (Intel core i5 2.4 Ghz, 4GB DDR3 RAM, Mac OSx v10.9.4) with a VM Linux Ubuntu 13.04 (64 bits, 512Mb RAM, 1 CPUs, 8GB HD). The virtual machine



(a) Delay for L1 (s1-s2)



(b) Delay for L2 (s2-s3)

Figure 8.5: Delay measurement for L1 and L2

Table 8.2: Summary evaluation of the CPU and memory usage

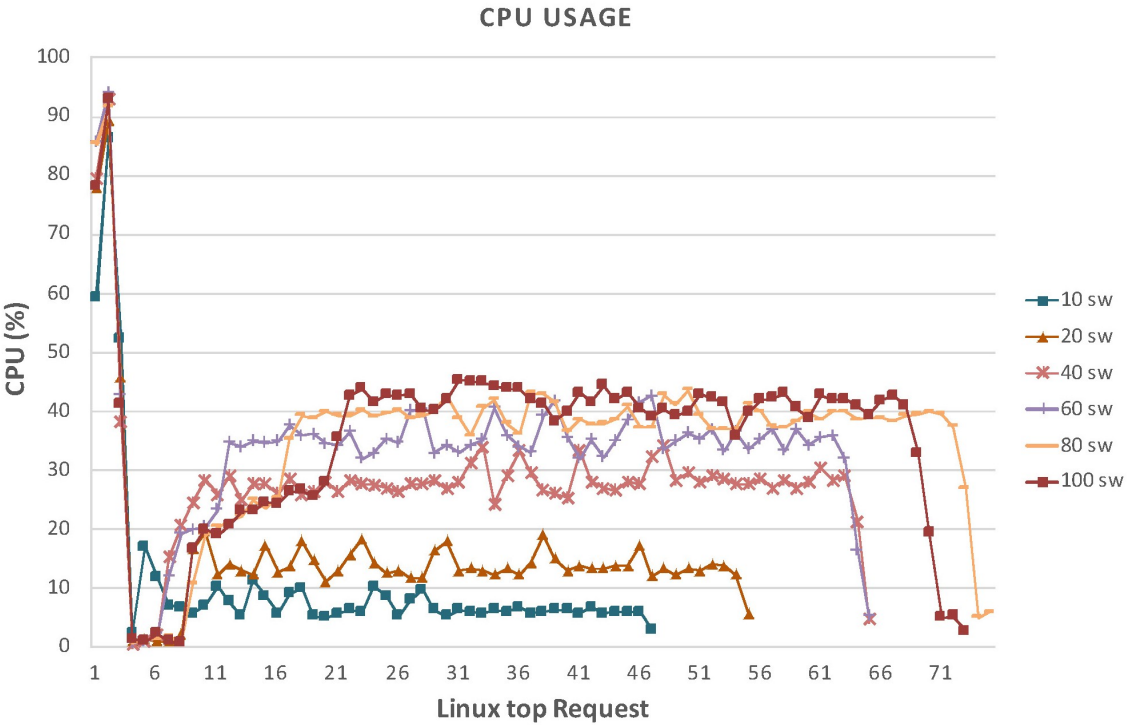
		CPU Usage (%)						Memory Usage (%)					
N	Req.#	10 sw	20 sw	40 sw	60 sw	80 sw	100 sw	10 sw	20 sw	40 sw	60 sw	80 sw	100 sw
1	1	59.3	77.9	79.5	85.6	85.4	78.1	1.9	1.8	1.9	1.9	1.9	1.9
2	8	6.7	2	20.6	19.3	1	0.7	10.3	9.9	10.3	10.2	9.9	9.8
3	15	8.7	17.3	27.6	34.6	23.6	24.6	10.3	10.4	10.5	10.4	10.5	10.5
4	22	6.3	15.6	28.3	36.5	39.3	42.6	10.4	10.6	10.7	10.5	10.8	11
5	29	6.3	16.3	28.3	32.9	39.9	40	10.5	10.7	11.1	10.7	10.9	11.4
6	36	6.7	12.3	33.3	34	36.2	43.9	10.5	10.8	11.3	10.9	11.4	11.4
7	43	5.7	13.3	27	32.3	37.9	44.4	10.6	10.9	11.3	11	11.4	11.4
8	50	-	13.3	29.6	36.3	43.6	39.9	-	11.2	11.3	11.1	11.4	11.4
9	57	-	-	27	36.9	37.3	42.2	-	-	11.3	11.3	11.4	11.5
10	64	-	-	21.3	16.3	38.6	40.8	-	-	11.3	11.3	11.5	11.6

executes the open source Mininet emulation tool 2.1.0 [LHM10] [GSB13]. Mininet enables the creation of custom topologies of OF-enabled switches, links and virtual host using Python scripts. Despite the fact that Mininet is currently one of the most used platforms to perform SDN experiments, the switching capacity in real time applications is limited by underlying host system capacity [min17]. Limited topologies with slower links provide and guarantee accuracy in network emulation in terms of timing accuracy and CPU/memory isolation.

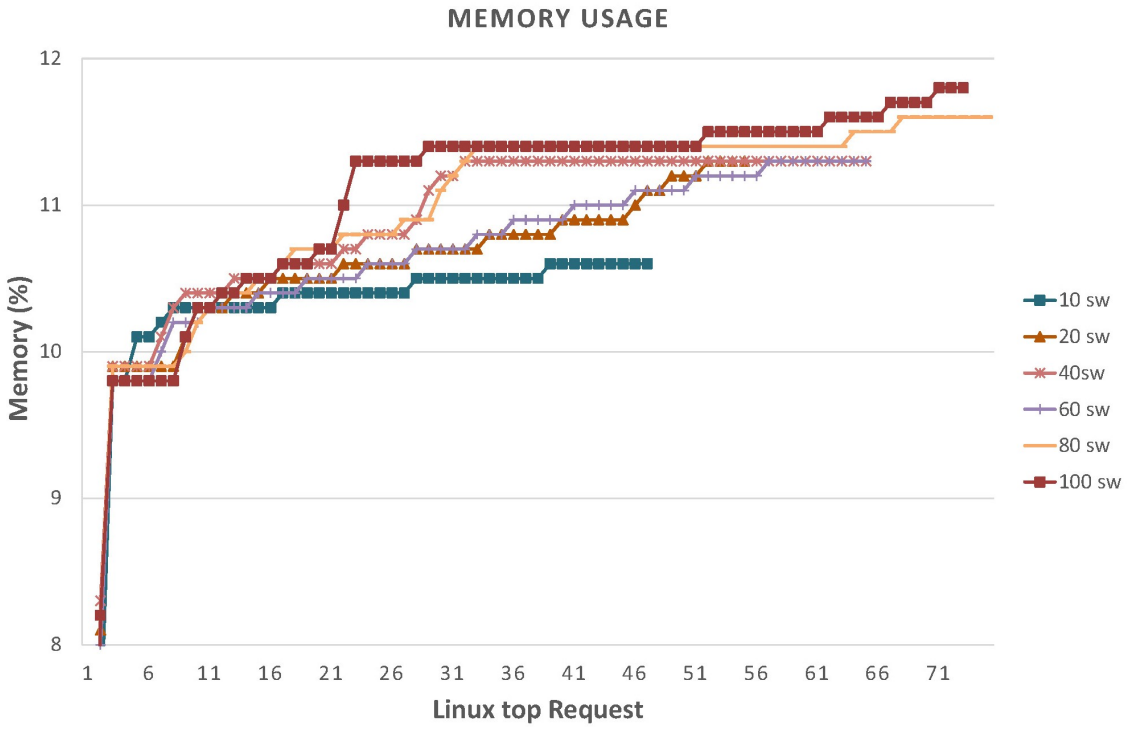
The test topology has four hosts (h1-h4), four switches (S1-S4) and has links between them (L1-L5). Two links (L1, L4) are assigned with a loss of 10 %. Mininet uses a netem linux kernel component (part of iproute2 package) to emulate an independent loss probability on the virtual links. In other words, 10% of the packets moving through links (S1 ->S2 & S2 ->S3) are randomly dropped.

This application uses the video file "highway_cif" [vid17] and the RTP/UDP streaming protocol using VLC server. The video is encoded in MPEG-1/2 (highway cif.ts) with a data size of 2.6 MB (2 572 968 bytes), resolution of 352x288, 2000 frames and a duration of 80 seconds using the fmpeg tool. The average bitrate of the video file is 0.25 mbit/s. However, the Ethernet frame and packet headers, RTP control messages and headers and additional information increases the real data rate of the network links.

In our experiments, the real data rate of a simple video stream is increased by about 0.4 mbit/s. For this reason, we limited the links to 1Mbps and sent different flows of videos in order to saturate the network. The controller behavior is highly dependent on the selected NOS (Floodlight, NOX, Beacon, Opendaylight), each with its own characteristics. The NOS performance depends on programming language (C++, Java, Python), the equipment, work load, topology and forwarding complexity [TGG+12]. In [VCBLGV13], we have presented an analysis of the performance of the different SDN controllers. The selected NOS is Floodlight version 0.90 that is also executed in the same VM. The performance of the controller is measured using Cbench tool [cbe17] in terms of throughput and latency as described in Table 8.3. The first row describes the Floodlight processing time of a switch request. In this case, the switch sends a request (82 byte size packet in)



(a) CPU Usage



(b) Memory Usage

Figure 8.6: CPU and memory usage

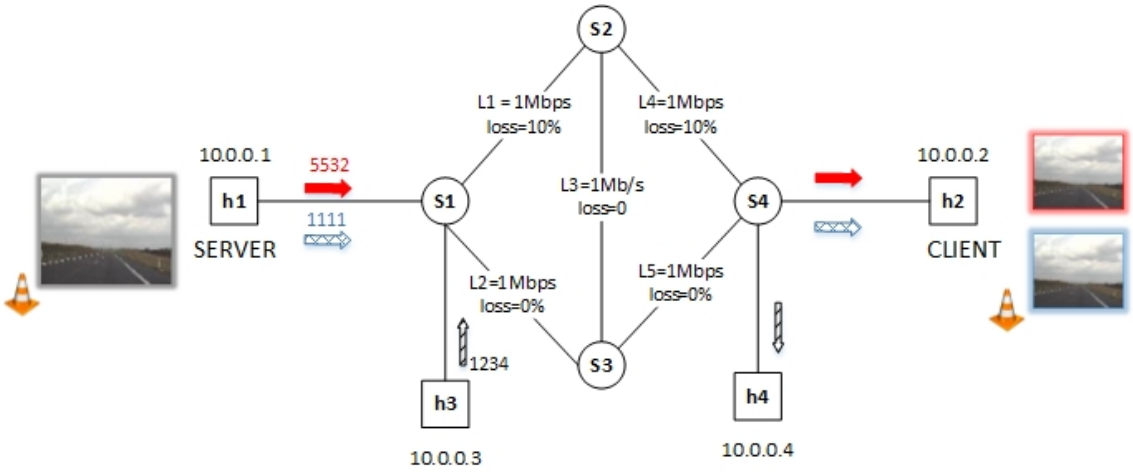


Figure 8.7: Optimized QoS test topology

and waits to the corresponding controller response. For its part, the second row shows the requests rate that a controller can handle. For that, the switch attempts to saturate the controller sending as many multi outstanding requests as buffering allows. In the experiment scenario, assuming that all the switches have an empty flow table and send all received PACKET IN messages to the controller (worst case scenario), the total requests number is about 914 request/second. This corresponds to the 7% of the average Floodlight capacity. As a result, the experiment follows the recommendations to minimize mutual interference between the processes (Mininet, Floodlight, VLC server and client, fmpeg) and maximize the accuracy of the results.

Table 8.3: Performance analysis of the floodlight controller

Measure	min.	max.	avg.	std. dev.
Latency (ms)	0.161	0.895	0.315	0.604
Throughput (request/s)	421.33	21 975.66	13 558.46	7 432.06

The experiment uses three flows: the high priority flow, a normal priority flow, and a background flow. High priority flow uses the port 5532 to send the stream between h1 and h2 with high priority (w/QoS). For its part, the normal priority flow (w/oQoS) uses the port 1111 to send the stream. The background flow sends the stream between h2 and h4 to saturate the links. During each run, 2000 frames were sent to each flow with an average rate of 25 frames/s. The background flow was introduced after 750 frames (t=30s). The samples were periodically monitored every 125 frames (5s). The streams are saved in different video files. Then, the files are decompressed as .yuv files and the PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index Metric) between source and destination is calculated using the Evalvid tool [vid17] [KRW03]. The PSNR measures the ratio between the original and the error signals of the video, while SSIM analyzes the perceptual distortion of the two fonts. Mininet as network emulator (data plane) uses the linux kernel and lightweight virtualization to emulate and run virtual host, links and software based network elements with a behavior similar to discrete hardware elements.

Similarly, the Floodlight controller is an SDN Controller that Works with physical and virtual OpenFlow enabled switches. In this experiment, the Floodlight app and mininet are separate processes which use OpenFlow (TCP port 6633) to interchange information between them (as a real network). Furthermore, the VLC streaming server and client are executed in separated processes running in virtual hosts (in the same VM). This causes slight variations between tests depending of the available CPU and memory resources. For this reason, we perform a basic Monte-Carlo method, that is, repeat the test scenario several times (20) and evaluate the corresponding average. However, in our experiments, the results of a single test present the same pattern, trends and similar conclusions.

The controller will assign different paths to the flows, even though they have identical (src ip, dst ip) tuple. In the first case (w/oQoS), the route is chosen based only on the minimum hop count. In the second case (w/QoS), the controller identifies the load and loss in the different links. With this information, the controller intends to find the best route to the flow. The algorithm assigns the path S1->S2->S4 as (w/oQoS). For its part, the traffic assigned as (w/QoS) avoids load and loss choosing the path S1->S3->S4. Furthermore, the controller configures the switches to process (w/QoS) packets with high priority. The accuracy of measured values by the network performance module depends on the monitoring interval. However, it is also verified that a reduction of period increases the CPU load of floodlight process and the number of requests on the switches. The network performance module is configured with a period monitoring time of 5 seconds.

8.2.2 Results

In order to reduce distortions, we display the trend line with an average of 25 frames. The results of the experiments are depicted in Figure 8.8, 8.9, 8.10 and 8.11. The red line (solid) represents the high priority flow (w/QoS) and the black line (dotted line) represents the normal priority flow (w/oQoS). Figure 8.8 and 8.10 show the PSNR and SSIM against the number of frames with a factor adjustment of $\propto = 0.5$ (worst case). For its part, Figure 8.9 and 8.11 are the results for a factor adjustment of $\propto = 0.75$ (best case). As expected, during the transmission period, the traffic with QoS clearly show better levels of efficiency compared with the w/oQoS traffic. For the case of $\propto = 0.5$, the average PSNR for w/ QoS is 27.47 dB against 26.00 dB of w/oQoS and the average SSIM is 0.81 against 0.79 accordingly. For the case of $\propto = 0.75$, the PSNR value for w/QoS is 30.41 dB against 26.18 dB of w/oQoS and the SSIM 0.85 against 0.80. However, Figures 8.8 and 8.9 present several troughs in cash plot. This effect is mainly caused by the fast moving objects and dynamic events in video scenes. During troughs, the video has an increase in bits per frame and data rate up to 0.302 mbit/s (increase of 21% of average video content bitrate). This increase, in turn, causes major congestion in links and reduces the measured quality perception.

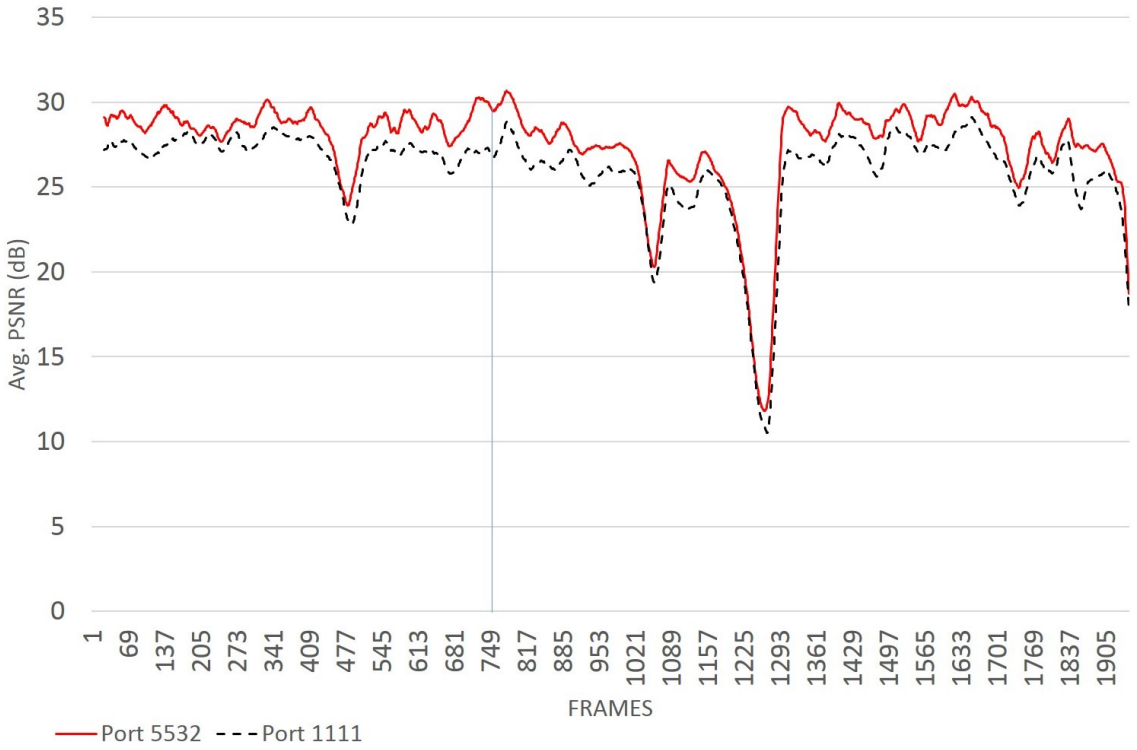


Figure 8.8: PSNR for $\propto$ =0.5

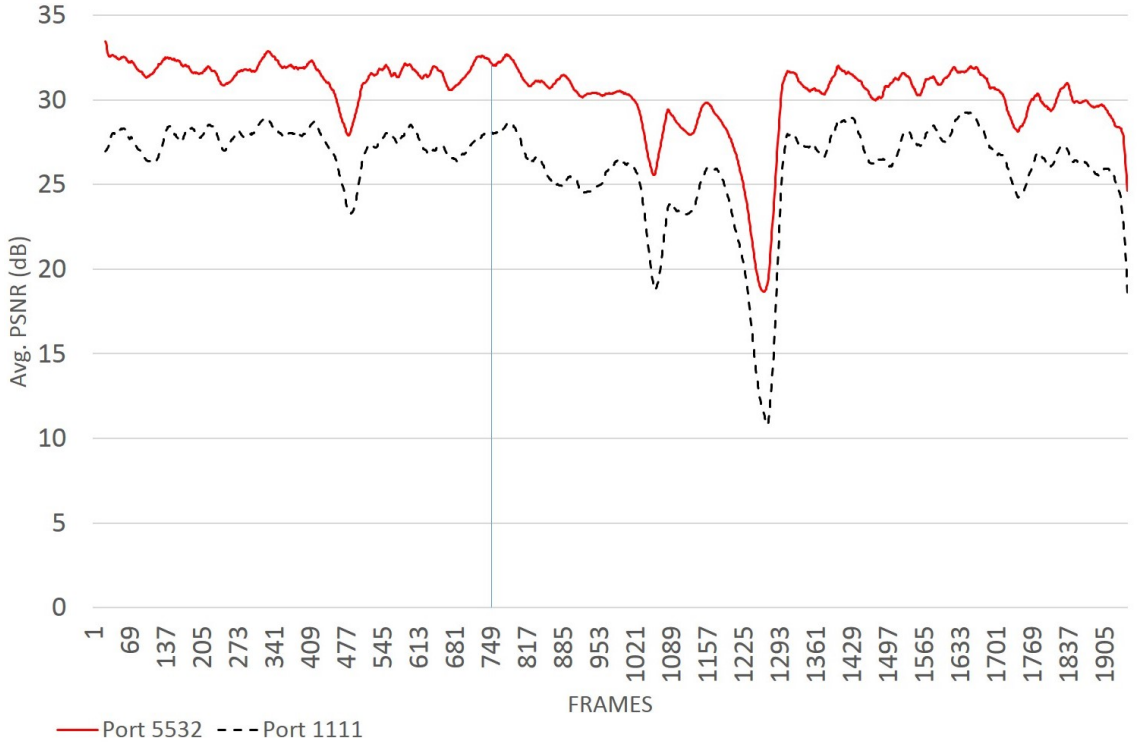


Figure 8.9: PSNR for $\propto$ =0.75

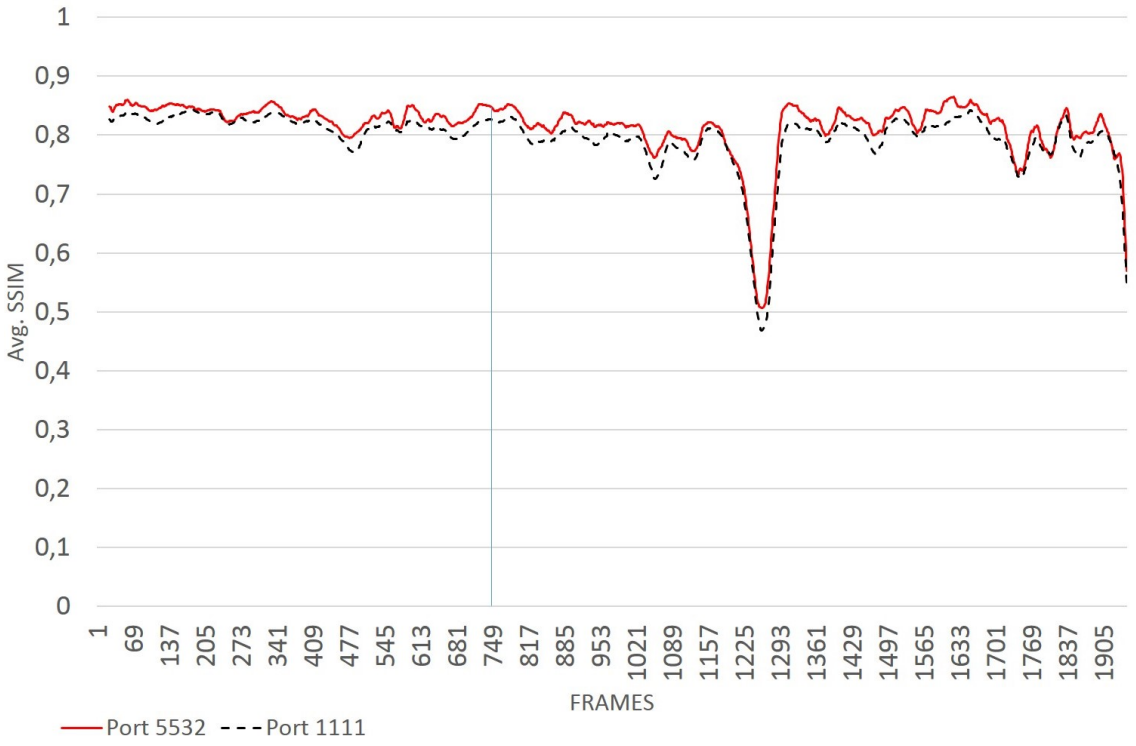


Figure 8.10: SSIM for $\propto$ =0.5

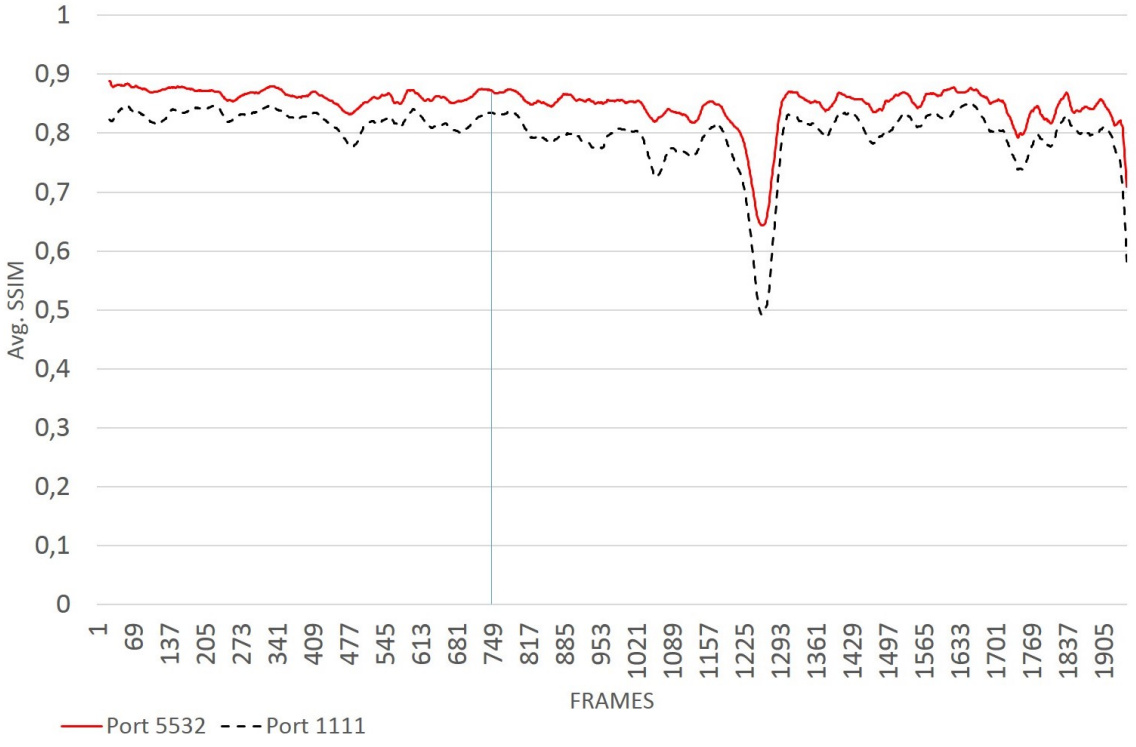


Figure 8.11: SSIM for $\propto$ =0.75

In order to summarize the results of the experiments, the MOS is taken into account. The difference with respect to the factors mentioned above (PSNR, SSIM) is that the MOS (ITU recommendation P.800 [itu96]) represents a subjective value of the user's perceptions with respect to a service. In other words, the MOS represents a QoE metric which estimates the grade of acceptability of the user. It proposes a scale of perception of 5 points: (5) excellent, (4) good, (3) fair, (2) poor and (1) bad. The procedure to calculate the MOS is based on relation scales with QoS metrics and is explained in [KRW03]. Figure 8.12 represents the average MOS of the two streaming in function of $\propto$. In this analysis, the perception of w/QoS streaming is in all cases better as w/o QoS and on average near or above 3 (fair).

We have performed additional tests using similar topology and sending only the high-priority flow. Even on excellent network conditions, the reception of the real time video streaming suffer variations caused by jitter, video decoder algorithm, CPU and memory resources of network and terminal device, among others. The obtained average MOS is 3.2 similar to the high priority MOS values of the experiments with additional flows (normal priority flow and background flow). This behavior demonstrates that the controller ensures QoS of high priority flow and balances other flows in the remaining available resources.

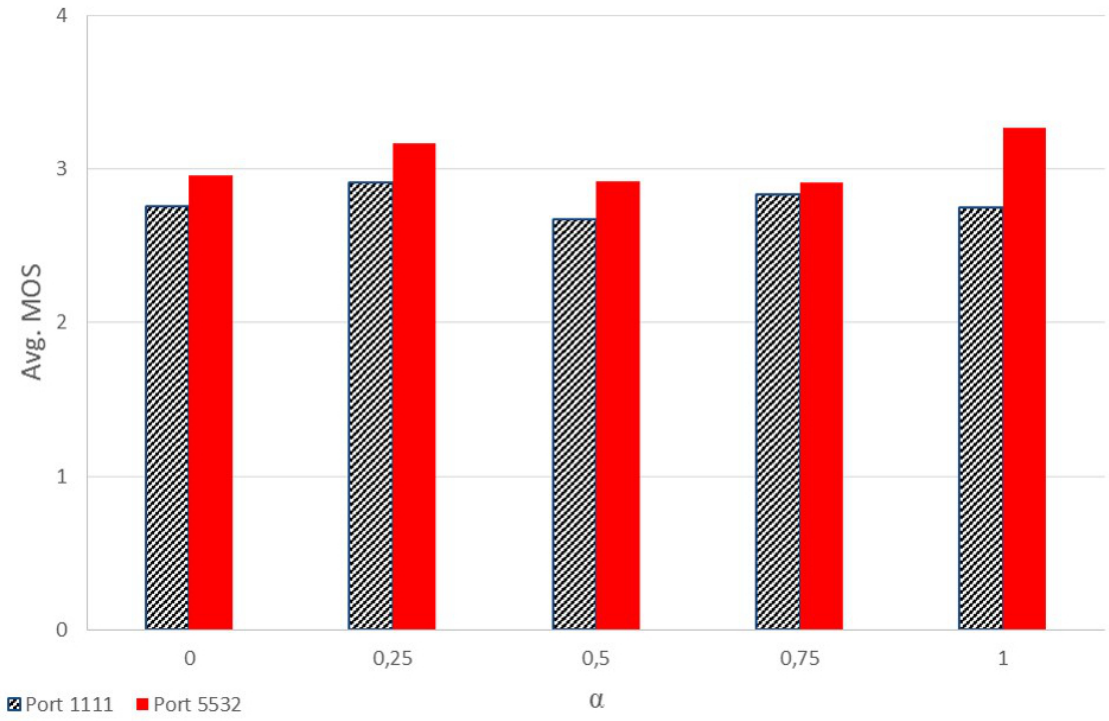


Figure 8.12: Results of MOS for the different values of $\propto$

8.3 Summary

This chapter presents the results of the experiments using an emulation tool and a sample topology in order to demonstrate the advantages of our frameworks. The results of experiments using an open source OpenFlow controller and a network emulator to monitor the send data rate, loss rate and delay confirm the feasibility of the monitoring framework. Similarly, the results in terms of PSNR, SSIM and MOS for a RTP/UDP streaming service demonstrate that the SDN controller ensures Quality of Service for a high priority real streaming flow in comparison with typical best effort engine.

Chapter 9

SELFNET Monitoring on SDN/NFV Self-Organized Networks

This chapter presents the SELFNET project. This design takes into account the scalability and flexibility requirements needed by 5G infrastructures. In this context, the present framework focuses on gathering and storing the information (low-level metrics) related to physical and virtual devices, cloud environments, flow metrics, SDN traffic and sensors. Similarly, the proposed Framework provides the monitoring data as a generic information source in order to allow the correlation and aggregation tasks. Our design enables the collection and storing of information provided by all the underlying SELFNET sublayers, including the dynamically onboarded and instantiated SDN/NFV Apps, also known as SELFNET sensors.

The rest of this chapter is outlined as follows: Section 9.1 introduces this chapter. Section 9.2 describes the SELFNET Monitoring and Discovery framework. Section 9.3 defines the Monitoring and Discovery Workflows. In the same way, Section 9.4 gives details on the implementation of the framework. Finally, Section 9.5 summarizes this chapter.

9.1 Introduction

SELFNET project has the potential to monitor the network status and service performance in order to provide intelligent and advanced capabilities. In this context, the SELFNET Monitoring and Discovery Framework is able to collect data from five different sources. It focuses on gathering and storing the information (low level metrics) related to physical and virtual devices, cloud environments, flow metrics, SDN traffic and sensor data.

SELFNET Monitoring and Discovery has additional advantages in comparison to current network monitoring solutions on SDN/NFV. The work proposed in [YCY+15] uses SDN to virtualize a switch to monitor traffic without the need for port mirroring. However, the system does not collect additional metrics at different levels (e.g. virtual,

sensors). Similarly, NetAnlytics [LTRW16] can efficiently monitor data plane packet flows and uses NFV to instantiate the network functions. However, the collected data is focused on flow metrics and does not provide customization of heterogeneous sources. The monitoring framework presented on [GKM+16] enables the Virtual Infrastructure Manager monitoring. It includes different agents able to collect data from VNFs. However, the monitoring framework is considered as internal functional block of the VIM and does not include additional sources (e.g. physical metrics) and customization of tasks (e.g. aggregation). In [SWL+16], the authors propose a reference framework for traffic engineering for SDN networks. The framework is composed of traffic measurement (network level) and traffic management (QoS, load balancing). However, the proposal does not include the customization capabilities provided by NFV engine.

SELFNET Monitoring and Discovery lays the SDN/NFV foundations to provide a contextaware model based on a customizable set of low-level metrics. It is able to organize the collected information according to different criteria such as virtual instances by tenant, metrics by virtual instances, physical devices by network location, physical metrics, flow statistics by device and metrics by sensor. As a result, the proposed framework will facilitate not only the querying process of gathered information but also the management of large amounts of data from heterogeneous data sources. In a next stage, the data of each of these layers can be correlated to provide enhanced information of the network status such as the relation between virtual instances and their respective physical device, the physical and virtual instances where a sensor is running, information related to LTE devices, edges and locations.

9.2 SELFNET Monitoring and Discovery

One of the main challenges of the SELFNET infrastructure is the monitoring and discovery of the different metrics generated by the underlying virtualized infrastructure. The metrics collected can include low-level metrics, KPI and HoN in order to have a complete knowledge about the network status. However, unlike the traditional monitoring solutions, where the monitoring nodes and information provided are static, the SELFNET sensors are virtualized network functions that can be dynamically allocated in different sections of the network. Moreover, the SELFNET Monitoring and Discovery Framework should be able to monitor a large amount of low-level metrics from several data sources. In this context, Figure 9.1 shows the eight interfaces used for either collecting metrics or sending the results of analysis tasks.

The SELFNET Monitoring and Analyzer interacts with other sublayers through different APIs. Table 9.1 describes the interface name, source, destination and the provided information. On one hand, several sources send their corresponding information to be processed and analysed. On the other hand, the output of the analysis is sent to the autonomic sublayer and the status of the sublayer is delivered to the broker sublayer. In order to achieve the functionality established by the SELFNET architecture, the proposed framework takes into account different design principles and methodologies, as summarized in Table 9.2.

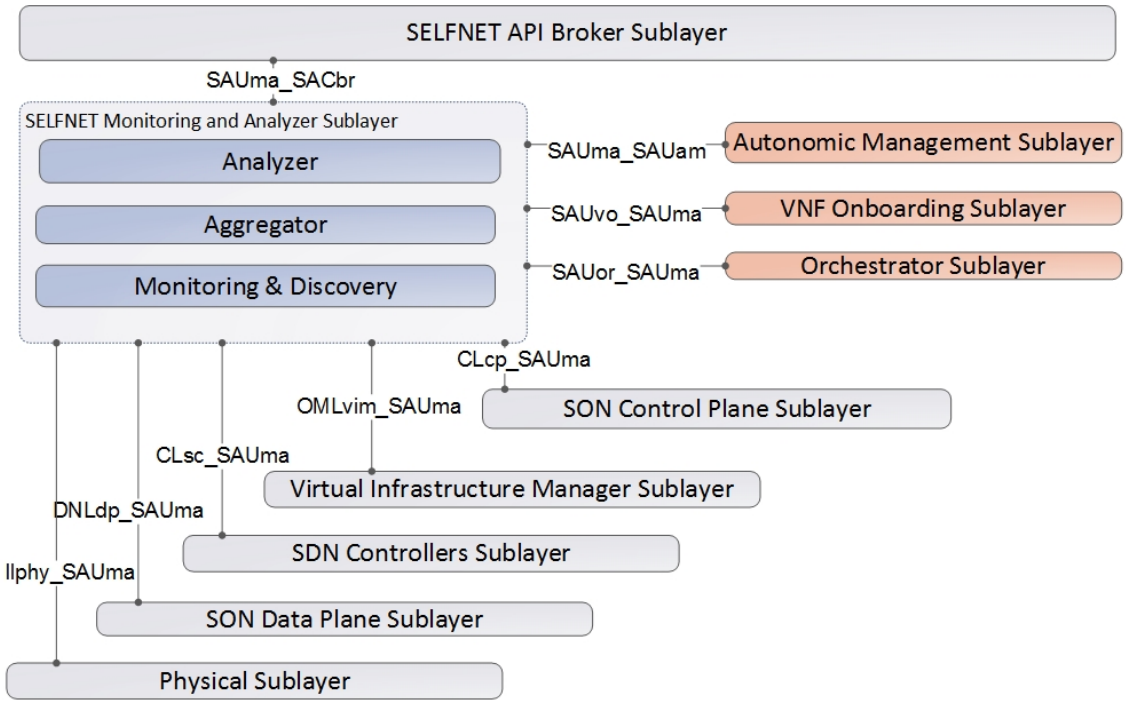


Figure 9.1: Monitoring and analyzer sublayer interfaces

Table 9.1: Monitor and analyzer sublayer interfaces

Interface Name	Source	Destination	Information
ILphy_SAUma	Physical	Monitor and Analyzer	Physical metrics
OMLvim_SAUma	VIM	Monitor and Analyzer	Virtual resources and metrics
DNLdp_SAUma	SON Data Plane	Monitor and Analyzer	Data Plane metrics
CLsc_SAUma	SDN Controllers	Monitor and Analyzer	SDN Controller metrics
SAUvo_SAUma	VNF Onboarding	Monitor and Analyzer	NFV Sensors description
SAUor_SAUma	Orchestration	Monitor and Analyzer	Instantiation of NFV Sensors
SAUma_SAUam	Monitor and Analyzer	Autonomic Management	High level summary of actual and predictive network issues
SAUma_SACbr	Monitor and Analyzer	API Broker	Status of Monitoring and Analyzer

9.2.1 High Level Architecture

The main goal of Monitoring and Analyzer sublayer is to provide a consistent set of components to analyze network status by the proper use of the monitoring information gathered from the running network infrastructure. With this objective in mind, the Monitoring and Analyzer Sublayer is divided in three sections: Monitoring and Discovery, Aggregation and Analyzer. The Analyzer Framework relies on HoN metrics to determine network status. The outcomes of this framework are sent to the Autonomic Management Sublayer. HoN metrics, in turn, are derived from the aggregated or correlated low-level metrics and events provided by the Aggregation Framework. Low-level metrics are collected in the Monitoring and Discovery Framework.

The whole architecture is divided in functional blocks, as shown in Figure 9.2. In

Table 9.2: Framework architectural requirements

Requirement	Description
Layered architecture	The framework follows a layered architecture, including a number of ordered and logically separated layers. In this way, the complexity on development process (design, implementation, evaluation) is reduced. Similarly, the interoperability between different vendors and technologies is supported.
Extensibility/Flexibility	The different layers and sublayers are composed using a modular design. The framework modularity offers advantages in terms of the usefulness in extensibility/augmentation. The open interfaces and APIs can be optimized by third parties to adapt their own solutions. Similarly, additional functionalities can be permanently or temporally integrated.
Multi-level scalability	The framework addresses network management concerns in large-scale networks. The monitoring sources are spread over virtualized network elements located strategically at regional or global levels. The framework enables elastic scalability employing the cloud computing engine.
Standards compliance	The design of the framework adheres to the standards that are considered relevant to the domain. It includes ETSI standards for NFV [ETS13a] and Open Networking Foundation (ONF) standards for SDN [PLHea11].

the bottom part, the underlying data sources are located. The Monitoring and Discovery framework introduces the concept of Data Source as a functional component that allows data to be transferred from the corresponding monitored element to the framework. A Data Source is capable of choosing the communication method, either polling or pushing, to gather or receive data from the monitored elements.

Polling and Pushing are main approaches used for distributed information dissemination [FP99], [LDC13]. The pull model is based on the request and response paradigm. In other words, the user initiates the data transfer and requests to the server for a specific piece of information, either synchronously or asynchronously. The server receives the request and responses back to the initiator with the requested information. Examples of the pushing approach include SNMP [CFSD90] or HTTP [FGM+99]. Meanwhile, the pushing method is based on the publish/subscribe/distribute paradigm. In this case, each agent (publisher) individually takes the initiative and sends the information to the subscribers through a distributed engine. The subscribers listen the set of channels based on their individual interest. Examples of pushing approaches include FCM [Dev17] and XMPP [SA04].

A Data Source implements the required interface to communicate with the monitored elements. For example, in order to receive data from a particular sensor, the Data Source may implement a server-side REST interface with a particular subset of methods exposed to the sensors, letting them send data to the monitoring framework. The Discovery function is related with the framework capabilities to detect the instance of new data sources and communicate them in order to collect metrics.

9.2.2 Virtual Sensors Descriptor

The Virtual Sensors Descriptor component is responsible for receiving, parsing and storing the information regarding onboarded sensor types available on SELFNET catalogue. This element interacts with two other SELFNET components: VNF Onboarding Sublayer and

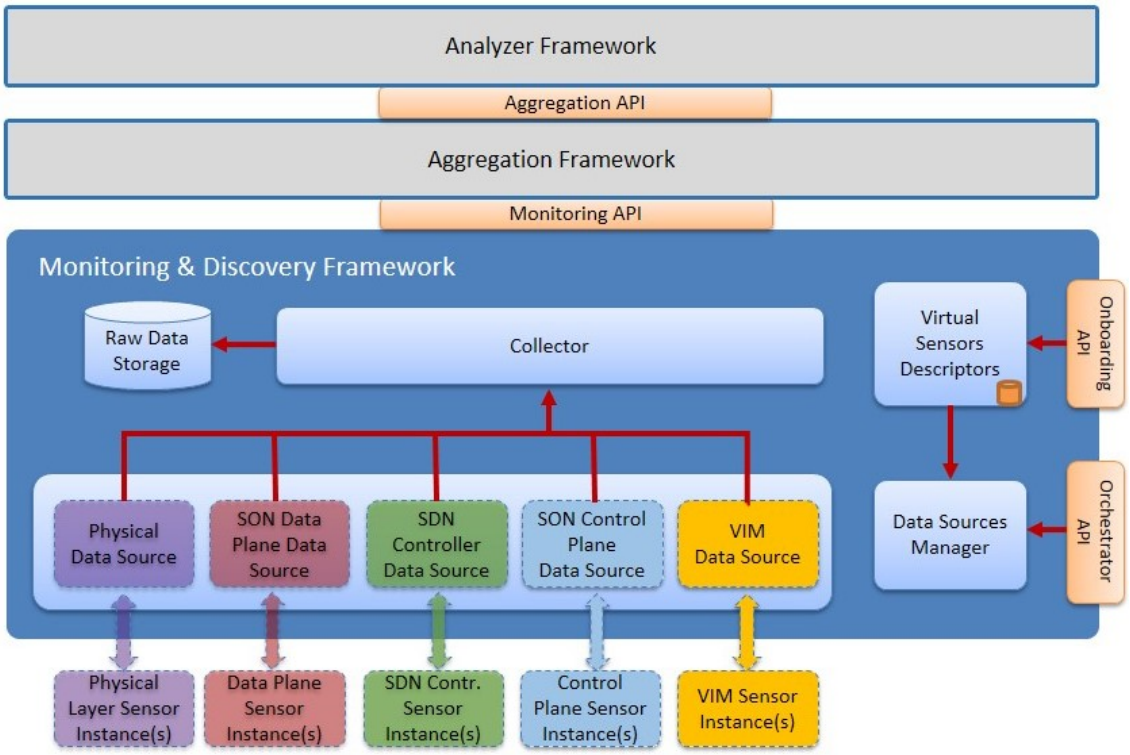


Figure 9.2: Monitoring and discovery framework architecture

Data Sources Manager.

The Virtual Sensors Descriptor uses the SAUvo_SAUma interface to allow the communication with the VNF Onboarding Sublayer. This interface uses a communication channel allowing VNF Onboarding Sublayer to report to Virtual Sensors Descriptors component whenever a sensor is onboarded, updated or removed in SELFNET catalogue. Every time an operation is performed in the SELFNET catalogue, the Virtual Sensor Descriptor needs to be notified in order to receive this information, parse it and update its internal sensor type database.

The VNF Onboarding Sublayer provides a message queue to broadcast any action performed on the catalogue, the Virtual Sensors Descriptors has a client that subscribes this channel and updates the local sensor catalogue. The local sensor catalogue is available to the Monitoring and Discovery framework via Virtual Sensor Descriptors - Data Sources Manager interface. This catalogue mirrors, from the VNF Onboarding Sublayer main catalogue, the essential sensor information needed to connect to a sensor and retrieve the sensed data. The catalogue does not have information regarding running sensor instances, but data related to the sensor metrics and communication.

9.2.3 Data Sources Manager

In the SELFNET Monitoring and Discovery Framework, the Data Source is defined as a functional component in the monitor side, which serves as an interface between the monitor and the monitored element and is in charge of collecting data from the

corresponding monitored device. For the instantiation of the data sources, the Data Sources Manager requires specific information (e.g. communication protocol, poll interval) about the sensor, which is obtained during the sensor onboarding phase. As soon as the sensor is instantiated, the Data Sources Manager is notified by the Orchestrator with the relevant instantiation information (e.g. IP address) from the sensor in order to correctly configure the data source. At this stage, the data source is ready to start collecting data.

9.2.4 Data Sources Instances

This section describes the different data sources and the way that the metrics are collected. As depicted in the Figure 9.3, the data collection shares a common structure. The source instances send the metrics to the corresponding data source. Then, the information is stored in the raw data through the collector.

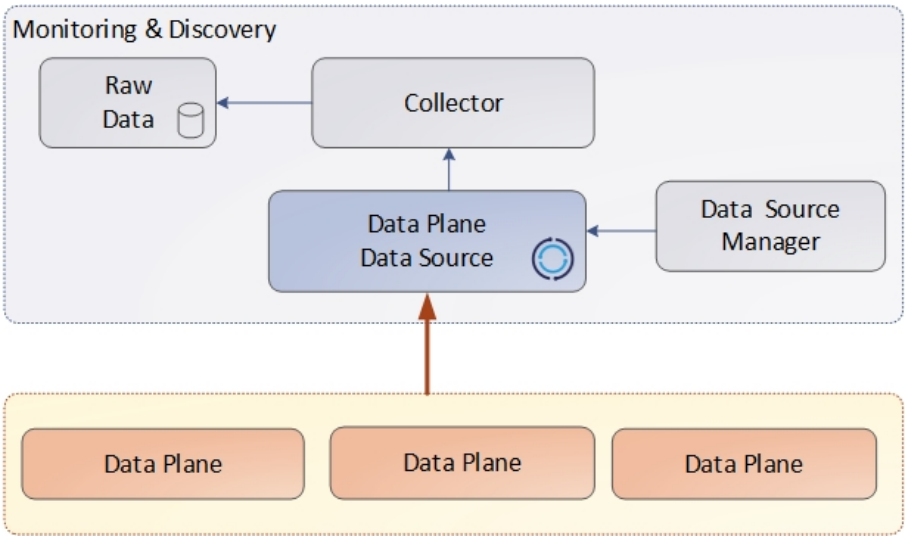


Figure 9.3: Data source

Table 9.3 summarizes the particular types of metrics depending on the source instances.

9.2.5 Collector

In the SELFNET monitoring framework, the Collector is defined as a functional component in the monitor side, which listens, on a message bus, for Data Sources metrics and events. Then, it transforms and publishes data into a Database and External Consumer Systems. The Collector uses a pluggable storage system, meaning that it can be changed by other databases if needed. This function occurs in multiple pipelines associated with Data Sources in order to provide configurable data flows for transformation and publication of data.

9.2.6 Raw Data Storage

Storing Metrics and Events offers different challenges. Metrics have a regular behaviour and events have an unpredictable nature. SELFNET will use more than one kind of

Table 9.3: Data source instances

Data Source Instance	Description
Physical Infrastructure	Metrics related with the physical equipment, which holds the virtual resources. It also includes the network elements that enable connectivity between the different components.
Virtual Infrastructure Manager	Metrics about the virtualized resources from the cloud environment that are running on top of the physical sublayer. It includes the tenants, virtual switching and routing management, and the location of compute resources attached to switch ports.
SDN Controller	It provides the information related with the network behaviour based on the view of the SDN Controller. In other words, the controller can infer the network statistics based on the information sent by the control plane - data plane interface (e.g. OpenFlow).
Data Plane	It provides low level network traffic characteristics. In this context, the flow level monitoring has become an appropriate solution. Flow level measurements can provide useful traffic statistics with small amounts of measured data.
SON Control Plane	It provides the collection of metrics from sensors that will be deployed through the entire virtual infrastructure. These sensors measure and collect specific physical and/or virtual metrics depending on the purpose of the sensor.

database to store events and metrics. Metrics will be stored in a *Time Series Database* (TSDB). This type of databases are optimized for handling time series data, arrays of numbers indexed by time (a datetime or a datetime range). Events will be stored in a NoSQL database whose main purpose is to store unstructured data for a short period of time. Raw data storage has the main goal to provide a stage where data can be queried for a short period of time. In addition, this storage can be used for more detailed analysis in order to take better decisions. It supports two kind of policies for metrics that need to be described on sensor descriptors. As an example, policies should look like: 1 day timespan for 1 minute granularity and 7 days timespan for 2 minutes granularity. This time aggregation is not intended to overlap features of the Aggregation Layer, it is only a way to support raw data for longer periods.

9.2.7 Northbound API

Monitoring and Discovery Framework exposes several APIs for other SELFNET components, but mainly for the Aggregation Framework to retrieve the available data. All stored data is published to a message bus, included in the Monitoring API, and can be reached by other SELFNET consumers. This should be the preferred method for forwarding data to the Aggregation layer. The information stored in the Raw Database can be accessed by a REST API. This API supports authentication and enables multiple operations: listing all metrics created, retrieving measures and filtering them over a time range, amongst others. It is also possible to query data for a specific granularity and for all available granularities. Optionally, data related to Events is stored in a Raw Database, but the preferred method to consume it is from message bus without any treatment or association logic. The database also exposes a REST API in order to query events that occurred in the near past.

9.3 Monitoring and Discovery Workflows

This section highlights the workflows across different components of Monitoring Framework and their relation with external components like the Onboarding or Orchestrator Sublayer.

9.3.1 Sensors Onboarding

Whenever a new sensor is onboarded to the architecture, several steps are carried out in order to inform other SELFNET components about the sensor functionalities and descriptors. This process is known as sensor onboarding and is depicted in Figure 9.4.

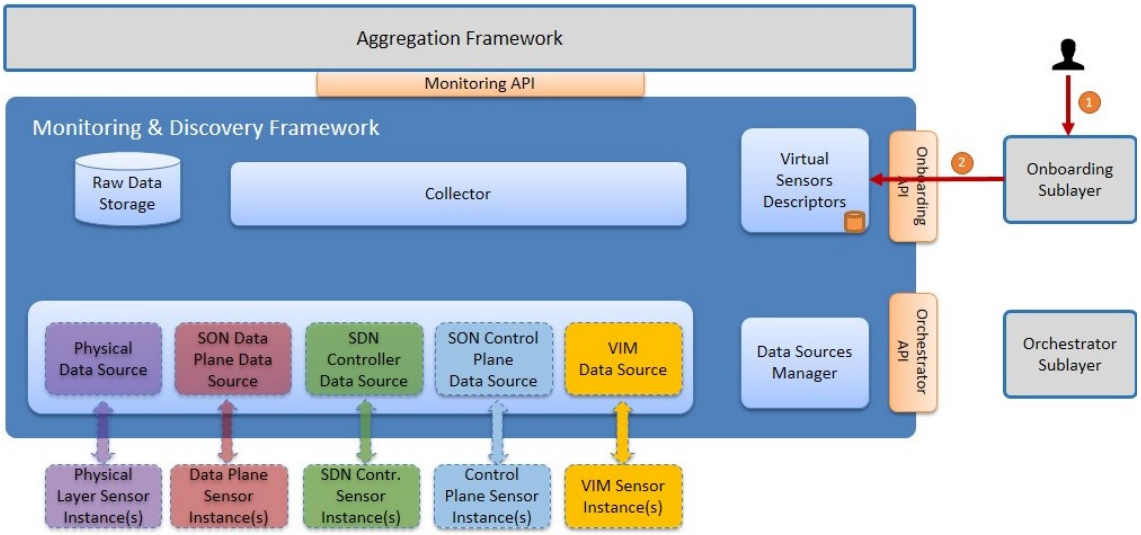


Figure 9.4: Sensor onboarding workflow

Two main components participate in this workflow, the Onboarding Sublayer and Virtual Sensors Descriptors, as is described in Table 9.4.

Table 9.4: Sensor onboarding workflow steps

Step	Source Component	Destination Component	Description
1	API Broker Sublayer	Onboarding Sublayer	A new virtual sensor is onboarded to the system either by using the SELFNET Graphical User Interface (GUI) or by an external system.
2	Onboarding Sublayer	Virtual Sensors Descriptors	The onboarding of a new sensor is published by the Onboarding Sublayer to a message bus to which the framework is subscribed by means the Onboarding API. The Virtual Sensors Descriptors takes care of collecting all the information needed to build its own representation of the sensor.

9.3.2 Sensor Instantiation

Sensor instantiation has a direct impact in Monitoring and Discovery Framework. Once instantiated, a sensor will start to produce data that will subsequently be available to the Monitoring and Discovery framework. For an effective communication between the sensor and the framework, a valid Data Source is required. Sensor instantiation workflow is depicted in Figure 9.5.

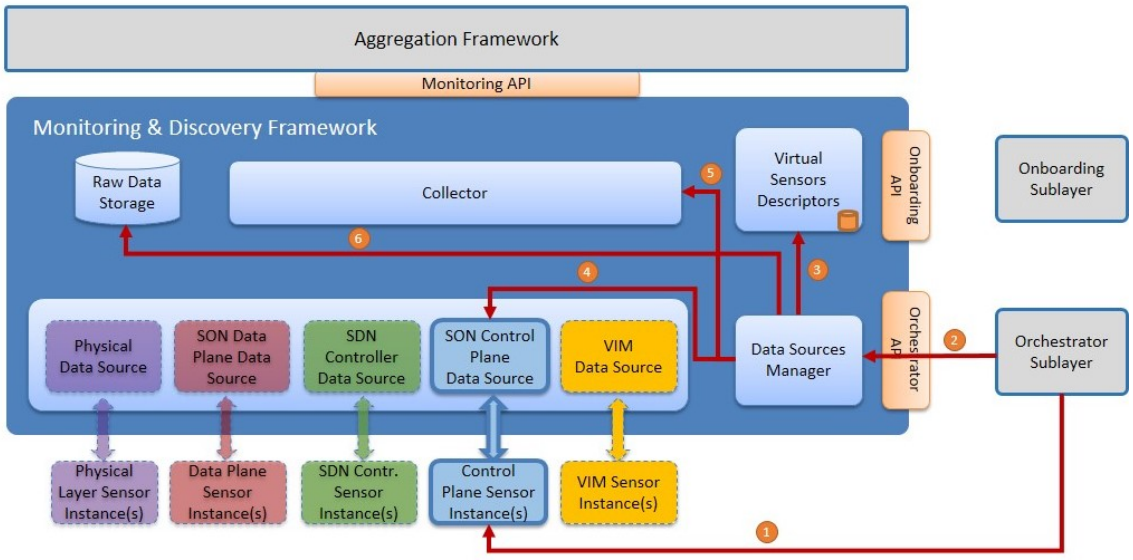


Figure 9.5: Sensor instantiation workflow

This workflow is described in Table 9.5 and it takes into account a Control Plane Sensor Instance. It is important to note that all sensors from the five sources follow the same process.

Table 9.5: Sensor instantiation workflow steps

Step	Source Component	Destination Component	Description
1	Orchestration Sublayer	Sensor Instance	A new virtual sensor (e.g. Control Plane) is instantiated by the Orchestration Sublayer (using the VIM).
2	Orchestration Sublayer	Data Sources Manager	When the sensor is deployed in the virtual infrastructure, the Orchestration Sublayer sends a notification about the successful instantiation that is received by the Data Sources Manager component. The notification message contains all sensor instantiation information (sensor type, IP address, port, tenant ID, etc.).
3	Data Sources Manager	Virtual Sensors Descriptor	When a new virtual sensor is instantiated, the Data Sources Manager requests the available metadata (full set of attributes related to a specific sensor, including the output parameters, sensing period, sensor type, etc.) from the Virtual Sensors Descriptors component.
4	Data Sources Manager	Data Sources Instances	Once the Data Source Manager receives the instantiated virtual sensor metadata from the Virtual Sensors Descriptor component, it creates and configures a new data source instance (in this example is the SON Control Plane Data Source) to be prepared to start collecting (listening or polling) data from the sensor. Onboarded information (metrics types, events types, communication protocol, poll periods, etc) and instantiation information (sensor type, IP address, port, tenant ID, etc.) are used by the Data Sources Manager to instantiate and configure the data source.
5	Data Sources Manager	Collector	The Data Sources Manager component provides the Collector component with the metrics and events that will be collected from the new virtual sensor, during the monitoring workflow.
6	Data Sources Manager	Raw Data Storage	Finally, the Data Sources Manager component provides to the Raw Data Storage component the new metrics (from the new virtual sensor) that should be stored and how (granularity of the stored metrics).

9.3.3 Sensors Instance Monitoring - Metrics and/or Events Workflow

After sensors are onboarded and instantiated in the virtual environment, data can start flowing, either pushed or polled, to the Monitoring and Discovery Framework. Figure 9.6 illustrates the sensors data flow (either metrics and/or events) towards the Monitoring and Discovery Framework.

The workflow steps are described in Table 9.6.

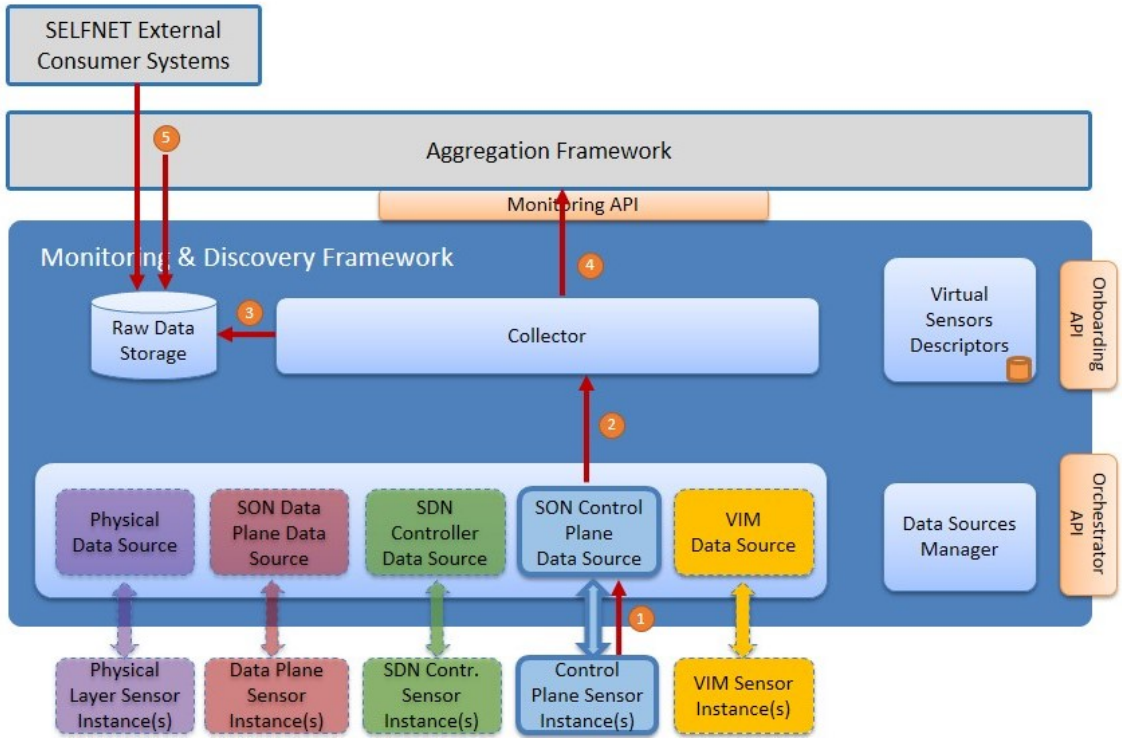


Figure 9.6: Sensor monitoring - metrics and/or events workflow

Table 9.6: Sensor monitoring - push metrics and/or events workflow steps

Step	Source Component	Destination Component	Description
1	Sensor Instance	Data Source Instance	Data is provided by the sensor. Two alternative mechanisms are provided: Push and Polling.
2	Data Source Instance	Collector	The Data Source Instance receives the raw metrics and/or events and provides these to the Collector (specifically, to the Collector Message Bus) using a common API.
3	Collector	Raw Data Storage	The Collector component transforms/ normalizes the received data and provides it to the Monitoring & Discovery Framework Raw Data Storage component.
4	Collector	Aggregation Framework	As in step 3, besides providing the received data to the Raw Data Storage component, the Collector component also delivers the collected data to the Aggregation Framework (using a message bus).
5	SELFNET external Consumer Systems	Raw Data Storage	Raw data (either metrics and/or events) can be retrieved/consumed by any SELFNET external system component using the exposed Raw Data Storage API.

9.3.4 Sensors Monitoring - Data Source Reconfiguration Workflow

During the sensor instance runtime, some specific parameters can be reconfigured without having to change the sensor instance. For example, one of the parameters that can be changed is the polling period. By default, the polling period is initially defined for each sensor type in the onboarded metadata. However, during the sensor runtime, this

parameter can be changed (for example as an autonomic management decision) without affecting the sensor-running instance. This workflow is illustrated in Figure 9.7.

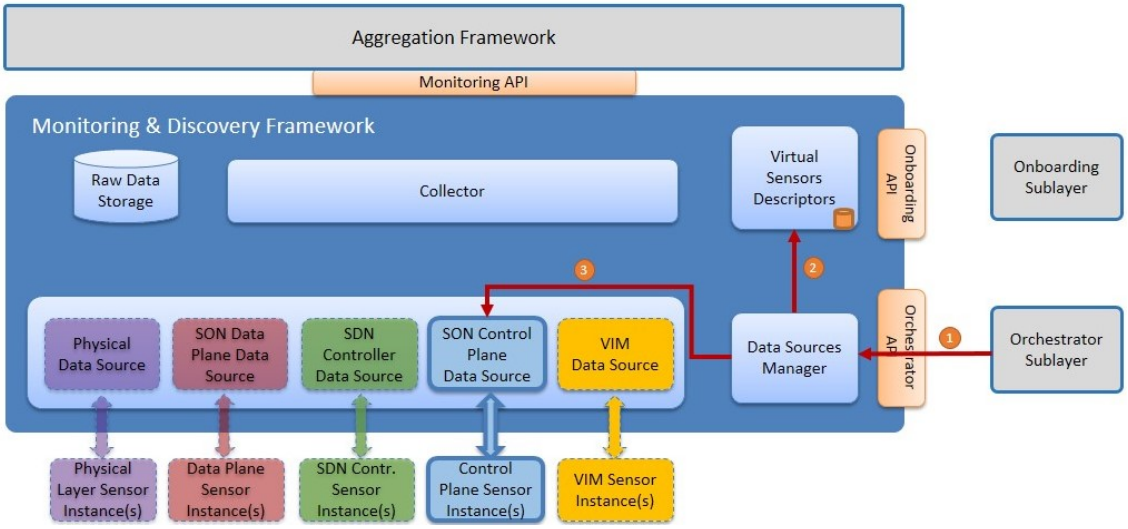


Figure 9.7: Sensor monitoring - data source reconfiguration workflow

The detailed steps of this workflow are described in Table 9.7.

Table 9.7: Sensor monitoring - poll metrics workflow steps

Step	Source Component	Destination Component	Description
1	Orchestrator Sublayer	Data Sources Manager	Orchestrator sublayer requests the Monitoring and Discovery Framework to modify the sensor instance polling period (e.g. due to an autonomic management decision).
2	Data Sources Manager	Virtual Sensors Descriptors	The Data Sources Manager requests the Virtual Sensors Descriptors for the sensor metadata information (if required).
3	Data Sources Manager	Data Source Instance	The Data Sources Manager requests the running data source instance to reconfigure the new polling period.

9.3.5 Sensors Instance Removal Workflow

Finally, when a sensor instance is removed from the infrastructure, the Data Source instance must be removed in the Monitoring and Discovery Framework. This workflow is illustrated in Figure 9.8.

The workflow steps related with the removal of a sensor instance are described in Table 9.8.

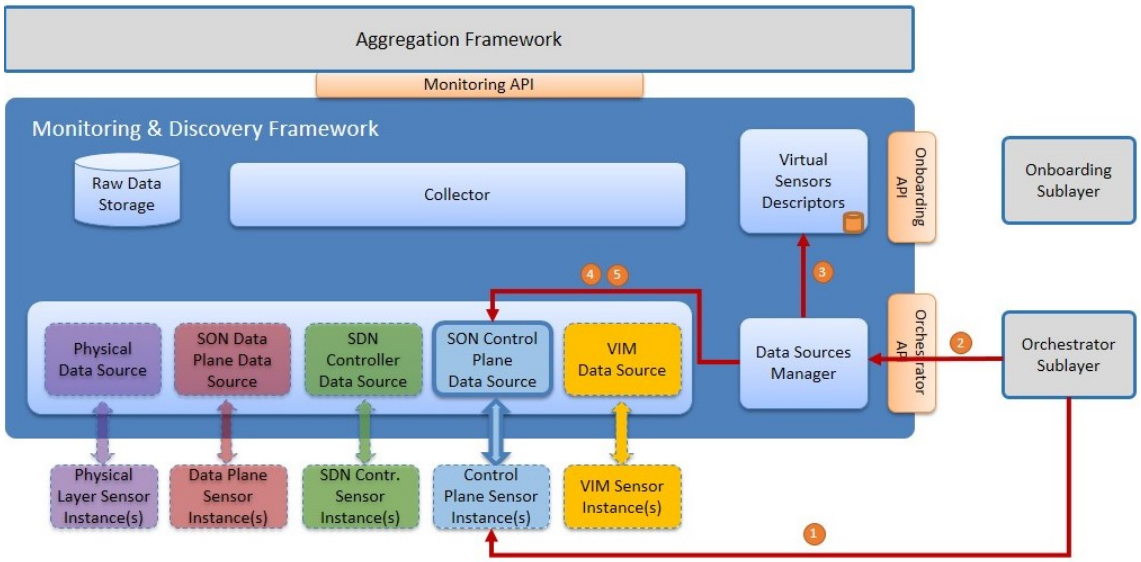


Figure 9.8: Sensor instance removal workflow

Table 9.8: Sensor instance removal workflow steps

Step	Source Component	Destination Component	Description
1	Orchestrator Sublayer	Sensor Instance	The Orchestrator Sublayer removes the running sensor instance (e.g. decision from the autonomic management system).
2	Orchestrator Sublayer	Data Sources Manager	The Orchestrator Sublayer notifies the Data Sources Manager that a specific sensor instance was removed.
3	Data Sources Manager	Virtual Sensors Descriptors	The Data Sources Manager requests the Virtual Sensors Descriptors for the sensor metadata information (if required).
4	Data Sources Manager	Data Source Instance	The Data Sources Manager requests the running data source instance to remove the configuration for the removed sensor instance.
5	Data Sources Manager	Data Source Instance	If the removed data sensor instance is the last instance of that data source type, the Data Sources Manager also removes the Data Sources Instance.

9.4 Implementation

The prototype implementation of the Monitoring and Discovery framework has several challenges in order to fulfill the above described requirements. The gathered metrics have several sources depending on the level of collection (physical, virtual, sensor). For this purpose, different open sources have been tested for implementation. Firstly, the Common Monitoring Frawework and how each component will interact in the monitoring process were described. Then, the implementation of Virtual Sensors Descriptors component, the implementation of each Data Source, the Data Sources Manager component, the Collector and the Raw Data Storage are detailed.

9.4.1 Common Monitoring Framework

In order to fulfill the Monitoring and Discovery requirements; we have chosen OpenStack project [SAE12] [ope17b] and its Telemetry service as the baseline to the framework implementation process. Openstack appears as a promised open source project to manage cloud platforms through a set of services (nova, neutron, keystone, telemetry, etc), which could be integrated not only with traditional networks, but also with SDN and NFV approaches. Similarly, the Telemetry Project [tel17a] was developed to facilitate the metering and monitoring of virtual resources from OpenStack deployments. This project manages different branches [tel17b] in order to provide alarming service (Aodh), data collection service (ceilometer/monasca) [cei17a], time-series database and resource indexing service (Gnocchi) [gno17]. Ceilometer proposes an architecture based on plugins that allows easy scalability, extensibility, meter/alarm customization and tracking of available resources by means the creation of new agents. A ceilometer agent collects not only information from OpenStack resources, such as Nova or Neutron [cei17a], but also external sources, such as LibreNMS tool [lib17] and ODL Time Series Data Repository (TSDR) [tsd17]. The Ceilometer architecture is shown in Figure 9.9.

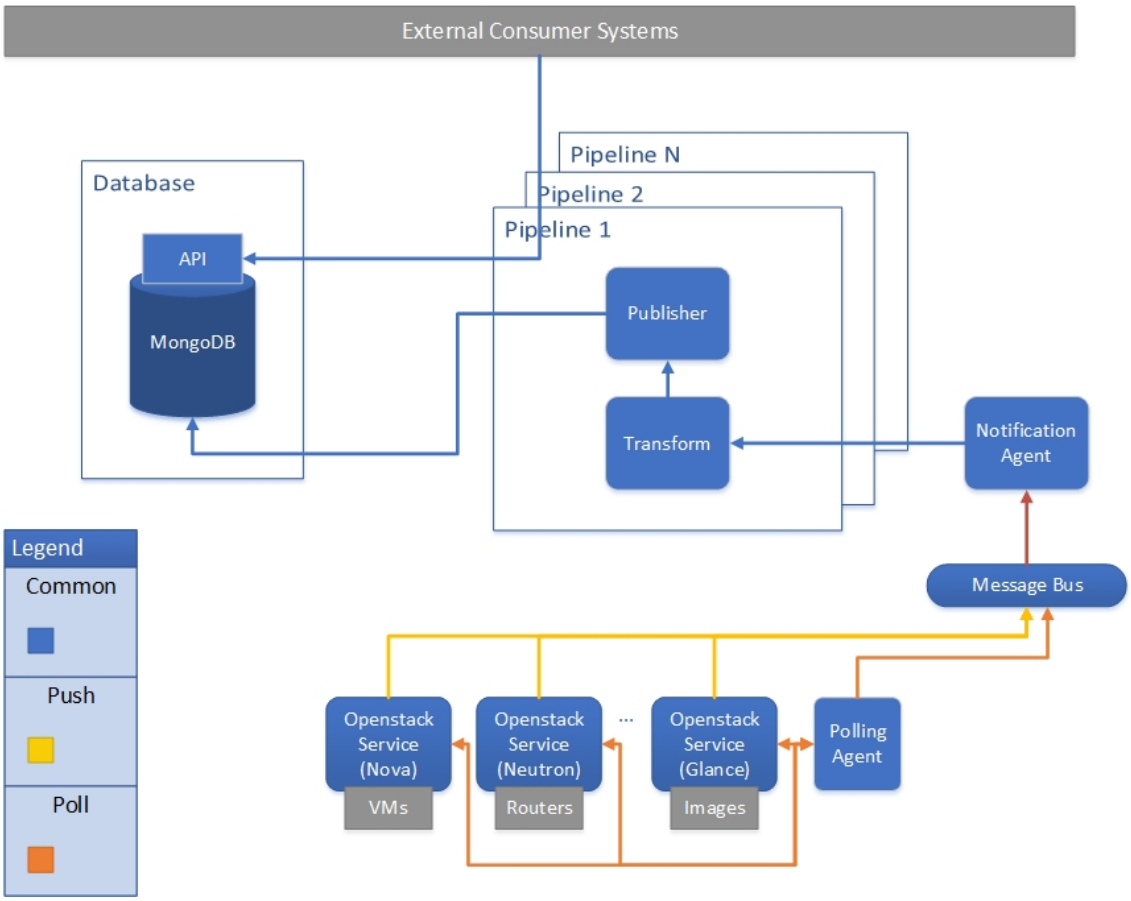


Figure 9.9: Ceilometer architecture

From a metric collector point of view, Ceilometer provides the polling and pushing methods for gathering data. The sources of information include not only OpenStack

services, but also additional customizable metrics. However, the increase in the number of sources and resources can decrease exponentially the system performance and cause scalability problems. To address this issue, Ceilosca project [cei17c] combines Ceilometer (telemetry collector) and Monasca projects [mon17]. Ceilosca is used as a metric storage tool that provides a fast API for data access. This approach deals with the above mentioned scalability issue [cei17b]. Figure 9.10 shows the mapping between SELFNET Monitoring Framework with Ceilosca approach and how it will operate.

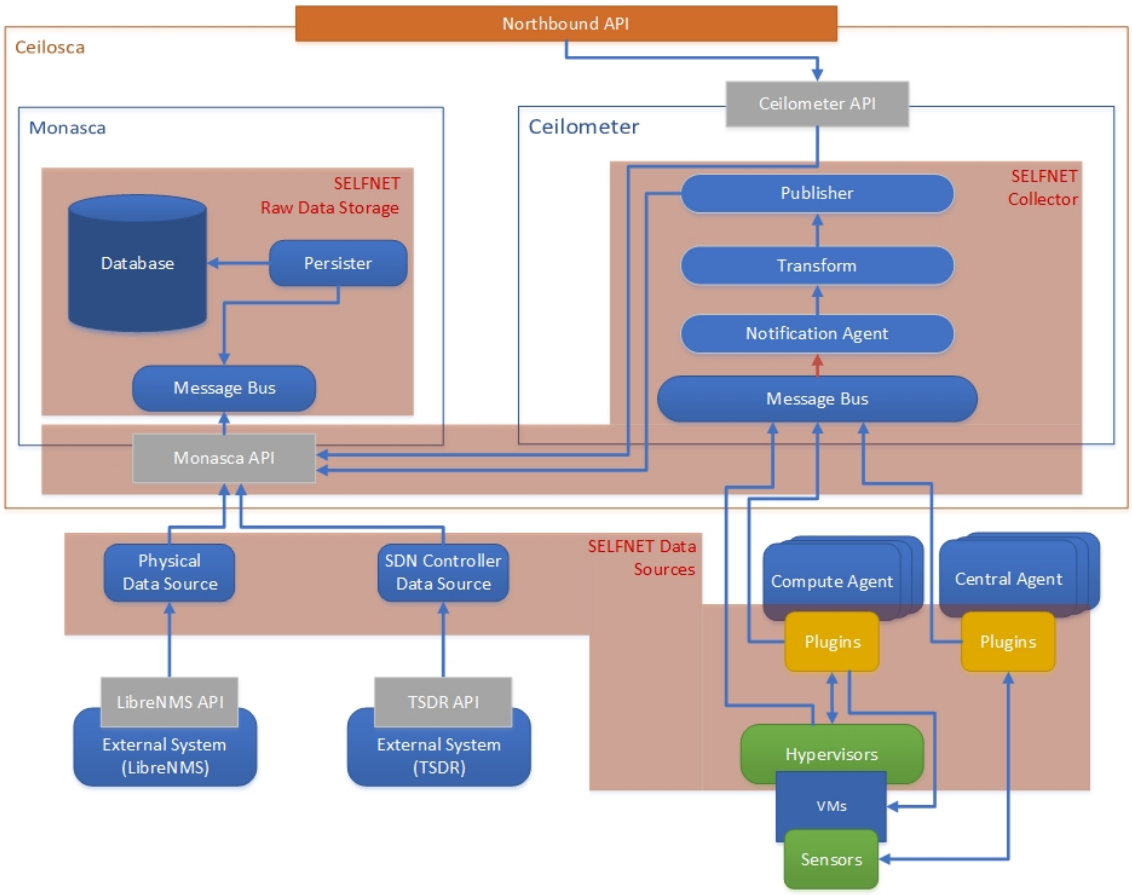


Figure 9.10: Mapping between ceilosca and SELFNET monitoring framework

The architecture has the following components:

- **Data Sources**. This component corresponds not only to traditional ceilometer data sources (Openstack services) but also to new plugins (for sensors to be developed). Data sources are related to the data gathering task. This component is further detailed in Section 9.2.4.

- **Collector**. This component is composed of the Notification Agent and Ceilometer Pipeline and the Monasca API. This component is further detailed in Section 9.2.5.

- **Raw Data Storage**. This component storages meters and events and it is related to the Ceilosca database. This component is further detailed in Section 9.2.6.

- **Northbound API**. It represents the API exposed by the Monitoring and Discovery Framework. It is composed of the Ceilometer API. The Section 9.2.7 gives more details about the API.

9.4.2 Virtual Sensors Descriptors

Virtual Sensors Descriptors are responsible for receiving, parsing and storing the information regarding onboarded sensor types available on SELFNET catalogue (Onborading Sublayer). For this purpose, it defines the next components:

- **Message Queue client**. To listen any action performed on the catalogue.
- **Local Database**. To store the sensor metrics and their kind of communication.
- **REST server**. Virtual Sensor Descriptors - Data Sources Manager interface

The Monitoring and Discovery framework listens for incoming messages provided by the Onboarding Sublayer when a change is made into the catalogue. The REST interface is developed using Python3 and a minimalist Web micro framework named Falcon. Since the sensor metadata is not relational, and it will only exhibit onboarded sensor data, the local database is a MongoDB database, a NoSQL database system. The interface with the VNF Onboarding Sublayer was developed also using Python3 and RabbitMQ, as the Message Bus.

The Virtual Sensors Descriptors expose a REST interface to the Data Sources Manager allowing it to reach the catalogue of sensor information. This interface needs to disclose a single endpoint to retrieve information. The client needs to provide the sensor type identifier in order to receive a JSON response with the information available on the local sensor catalogue. The Message Queue client is prepared to receive three type of actions:

- **Create**. Add a new monitoring piece of metadata to the local database.
- **Update**. Replace the monitoring metadata.
- **Delete**. Delete the stored metadata for a specific sensor.

Each collected metric is composed of a unique name, “metric-name” and a set of sample metrics, each one identified by a name, a resource which provides the origin of the value, a unit and a type (e.g., cumulative or delta). The Virtual Sensor Descriptors also exposes the sensor communication to the Data Sources Manager, composed of the protocol, a set of endpoints with the metric id and the endpoint value, the method (push or poll) and a list with metadata composed of a key value pair.

9.4.3 Data Sources Manager

To create the corresponding Data Sources in line with the sensor specifications (delivered data, protocol and period), the Data Sources Manager component is implemented. It dynamically creates specific Data Sources to map the type of sensor instantiated in the

infrastructure, establishing the communication channel to gather information. In order to receive information about sensor activities (instantiation, reconfiguration, delete), a RabbitMQ interface between the Data Sources Manager and the Orchestrator layer is implemented. Then, the information must be classified by the Action Selector component, as is illustrated in Figure 9.11. Action Selector is used to analyze the type of action performed by the Orchestrator through specific messages, identifying which operation is done (sensor deployment, remove or reconfiguration). According to the message obtained from the orchestrator, the action selector decides which operation is done and notifies the respective module.

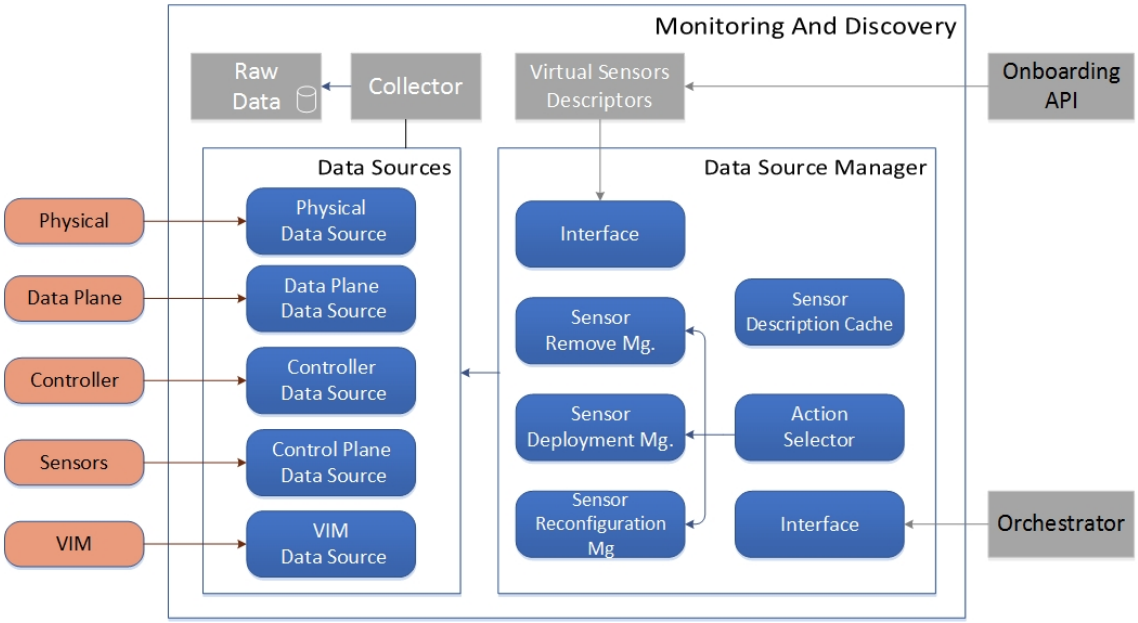


Figure 9.11: Data source manager: sensor operations

Three functional blocks are implemented: i) the sensor deployment manager, ii) the sensor remove manager, and iii) sensor re-configuration manager. Furthermore, an interface is implemented between Data Sources Manager and Virtual Sensors Descriptors component, which can retrieve the necessary information, such as the type and ID of SDN/NFV applications, the metrics collected by the sensor, the data transmission mode (poll/push), among others. The retrieved information is temporarily stored in the Cache, called Sensor Description Cache. The sensor description is used to manipulate the sensor during the sensor lifecycle.

9.4.4 Data Sources Instances

A Data Source is defined as an interface between the Monitoring and Discovery Framework and the monitored element. In this way, it is able to gather data from the corresponding monitored device. For this purpose, SELFNET framework take advantage of several open source monitoring tools, as described in Table 9.9.

Table 9.9: Open source dependences

Step	Source Component
Physical Infrastructure	LibreNMS [lib17]
SDN Controller	ODL [MVTG14]
Data Plane	ODL-TSDR [tsd17]
VIM	OpenStack [SAE12]
Control Plane Sensor	Ceilometer [cei17a]

9.4.5 Collector

Having in mind the mapping between SELFNET architecture and Ceilosca (Figure 9.10). The Collector is composed of the Monasca API and four Ceilometer components: the Message Bus, the Notification Agent, the Transform and the Publisher. The Notification Agent consumes data from the Message Bus, delivering it to the Transform, where the data is normalized. Then, the Publisher(s) publish the data into the destination: the Ceilosca database (by using the Monasca API) and external consumer systems (such as a message bus used by the Aggregation Framework). Moreover, the Monasca API can directly publish telemetry data into Monasca, without using Ceilometer, from non-Openstack services (e.g., LibreNMS, ODL-TSDR).

9.4.6 Raw Data Storage

Ceilometer provides several ways to store both meters and events. By default, Ceilometer stores meters and events in a MongoDB, a NoSQL database, where every record is stored in the database (raw data). As of the date, the Ceilometer project recommends to replace MongoDB by the TDBaaS Gnocchi (Time Series Database as a Service) to store meters, while keeping MongoDB for events storage, especially when facing low latency use cases. The main reason for this change is the scalability problems of MongoDB when stored data are queried (bad performance when increase the amount of records stored). SELFNET Monitoring Framework presents some alternatives to store the data. On one hand, Monasca, by default, stores data using InfluxDB (TDBaaS like Gnocchi). On the other hand, Ceilosca provides a faster and more complete API that enables, not only operations like aggregation, max, count, avg, over data; but also a way to insert non-telemetry data (external to Ceilometer).

9.4.7 Validation - Monitoring GUI

The Monitoring and Discovery Framework has been implemented in a virtual environment as a proof of concept, over a single physical server (ThinkServer TD350: Intel Xeon Serie E5-2600 v3 -16 cores, DDR4 512 GB - 2133 MHz). Six virtual machines have been deployed through VirtualBox hypervisor, as is shown in Figure 9.12. These VMs represents the Controller Node, Compute Node, Telemetry Node, Opendaylight Node, LibreNMS Node and the Monitoring Server.

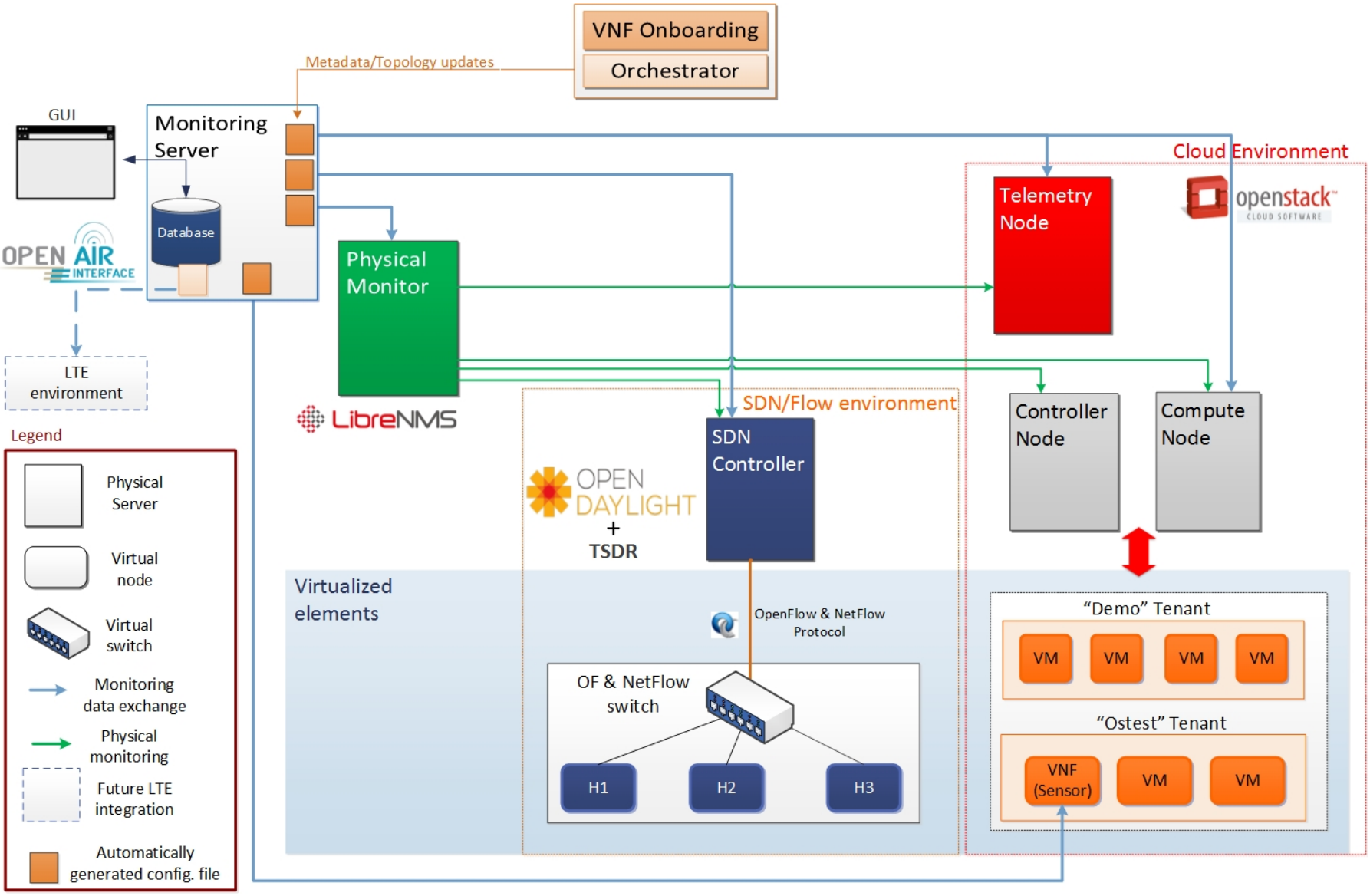


Figure 9.12: SELFNET monitoring framework testbed

The Monitoring Server retrieves the required information from data sources: Physical Monitor (LibreNMS), Cloud Environment (OpenStack), SDN/Flow environment (ODL+TSDR), LTE environment (OpenAir) and Sensors. The Monitoring Server manages the communication not only with the VNF Onboarding Sublayer and the Orchestrator Sublayer, but also with all data sources through a set of configuration files. Physical Monitor is able to collect data from SDN Controller, Compute Node, Telemetry and Controller Node, which emulates a Physical Server. In turn, Telemetry Node collects information from each OpenStack service available in the cloud environment. For its part, Monitoring Server retrieves sensor metrics through the Sensor API. Regarding to the collection of LTE metrics, this is part of the ongoing work. Moreover, the communication between these nodes, are done throught two networks, as follows:

1. The Management Network (MGNT) is in charge of the communication between Openstack, LibreNMS and ODL Servers.

2. The Public Network provides internet access and must be reachable by anybody. In particular, some OpenStack API (Neutron, Horizon) are exposed via this network.

Furthermore, two OpenStack services are mainly involved in the communication between virtual instances (VMs); i) the Nova Service that enables the instantiation of a new virtual machine and ii) the Neutron Service that is responsible of the management of the virtual networks.

In order to visualize the collected information, a simple Monitoring GUI was developed as a three-tier architecture based on a client-server approach.This GUI shows selected data from physical layer devices, virtual layer devices, LTE, sensors and flows level (Figure 9.13).

For example, the view of the physical layer displays a list of all discovered network devices and their most relevant parameters: device identification, hostname, location, port and MAC address. The Virtual Layer View shows the tenant list (list of tenant with their respective ID, name and description) and the device list (related information of each virtual instance) as is illustrated in Figure 9.14.

For its part, the Flow View describes information about the gathered flow metrics (OpenFlow and NetFlow) and the Sensor View displays a list of all deployed sensors and their most relevant parameters: sensor identification, IP Address, tenant identification, tenant network identification, location, type of sensor, and their values and statistics; as is illustrated in Figure 9.15.

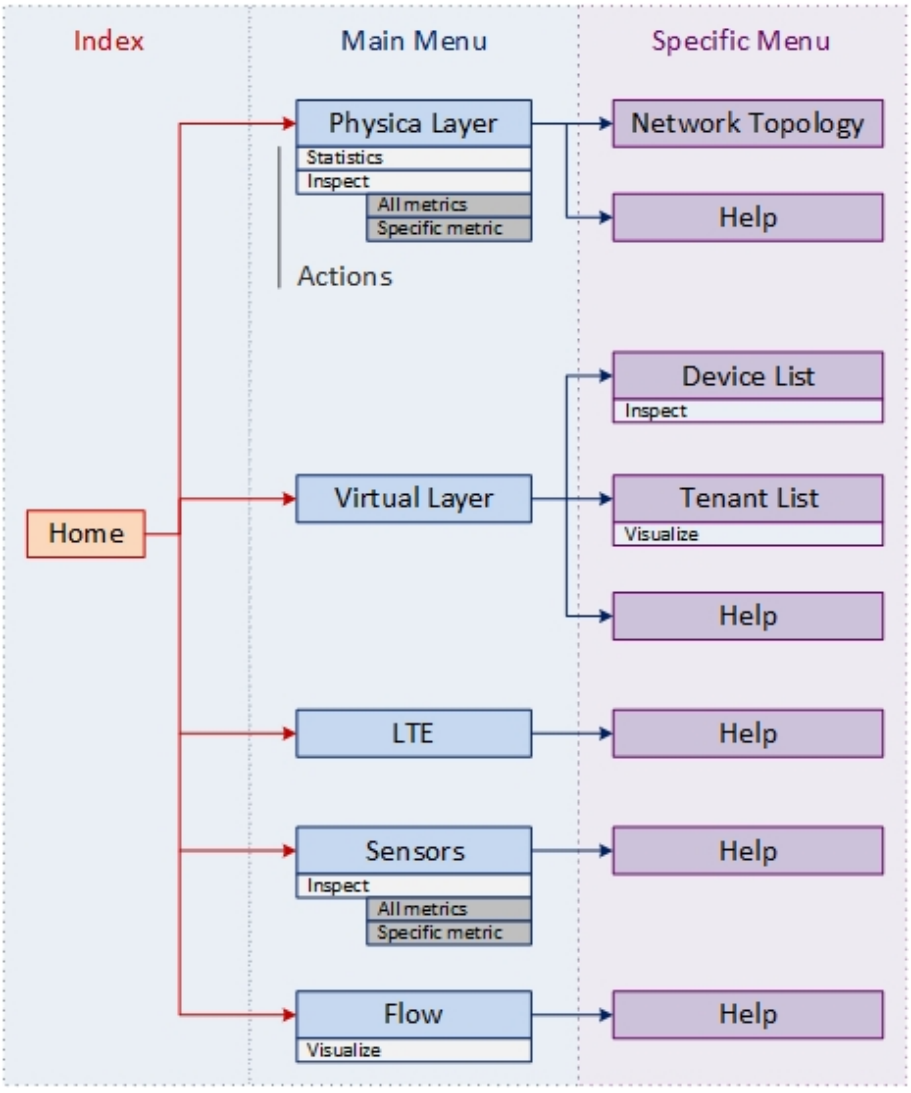
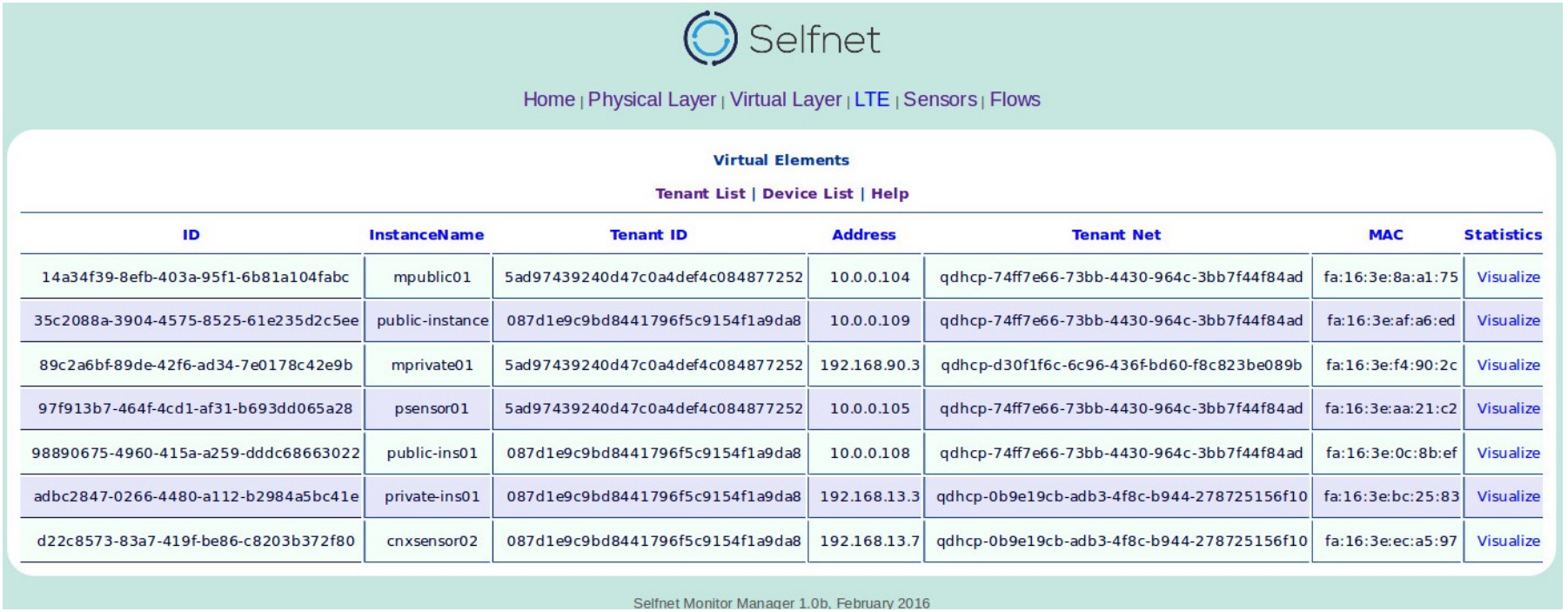


Figure 9.13: SELFNET Monitoring GUI navigability



ID	InstanceName	Tenant ID	Address	Tenant Net	MAC	Statistics
14a34f39-8efb-403a-95f1-6b81a104fabc	mpublic01	5ad97439240d47c0a4def4c084877252	10.0.0.104	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:8a:a1:75	Visualize
35c2088a-3904-4575-8525-61e235d2c5ee	public-instance	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.109	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:af:a6:ed	Visualize
89c2a6bf-89de-42f6-ad34-7e0178c42e9b	mprivate01	5ad97439240d47c0a4def4c084877252	192.168.90.3	qdhcp-d30f1f6c-6c96-436f-bd60-f8c823be089b	fa:16:3e:f4:90:2c	Visualize
97f913b7-464f-4cd1-af31-b693dd065a28	psensor01	5ad97439240d47c0a4def4c084877252	10.0.0.105	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:aa:21:c2	Visualize
98890675-4960-415a-a259-dddc68663022	public-ins01	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.108	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:0c:8b:ef	Visualize
adbc2847-0266-4480-a112-b2984a5bc41e	private-ins01	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.3	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:bc:25:83	Visualize
d22c8573-83a7-419f-be86-c8203b372f80	cnxsensor02	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.7	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:ec:a5:97	Visualize

Figure 9.14: List of SELFNET virtual elements on the virtual layer view

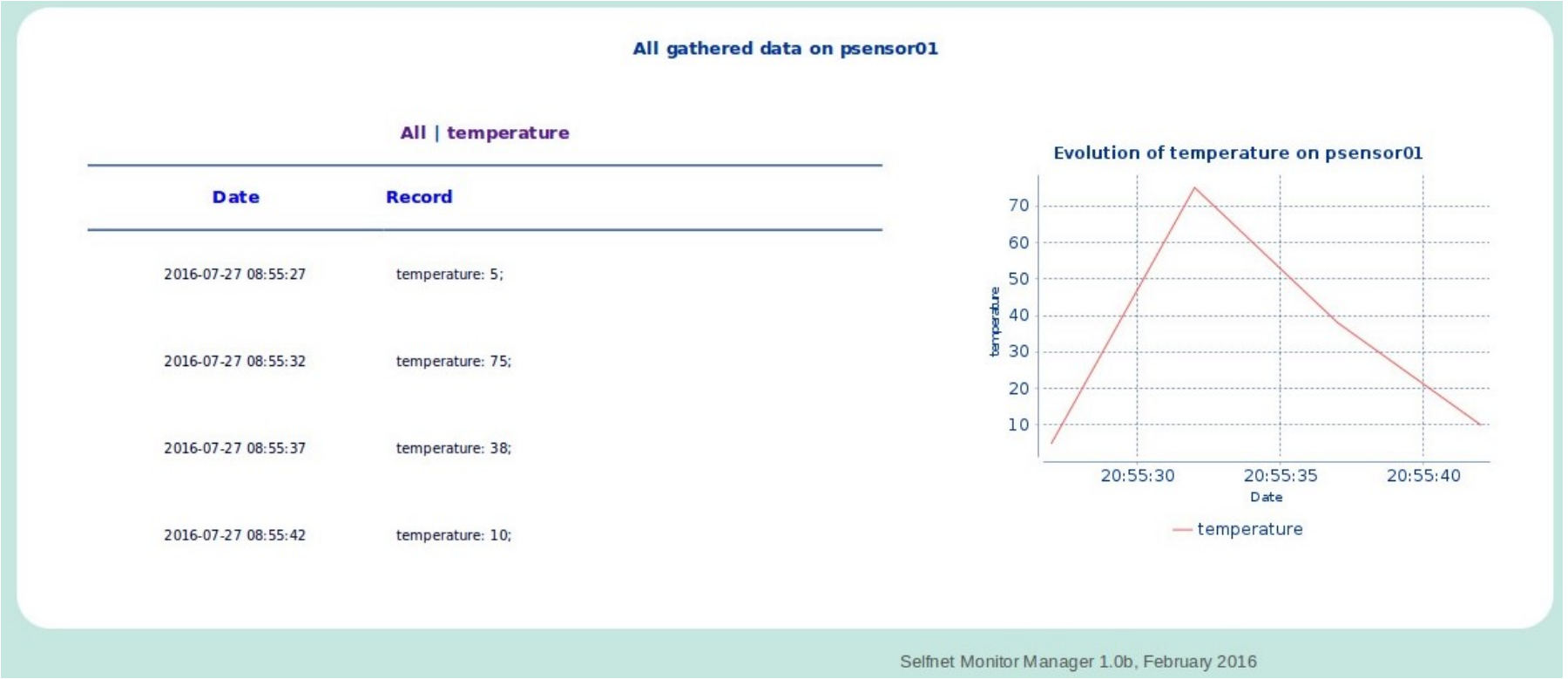


Figure 9.15: Details of information gathered by a SELFNET Sensor

It is important to note that "psensor01" is a simple temperature sensor. It is running in a virtual machine of a specific tenant (ostest), from which the data is retrieved every 5 seconds. The monitored metrics are defined in the Onboarding Sublayer and the instantiation information in the Orchestrator Sublayer. This example shows the temperature values for each sample and their evolution over time. This module is able to i) customize the monitored metrics by each sensor type and ii) gather these metrics from the respective data source.

An additional way in validating the capabilities of the framework includes the analysis of specific Use Cases. The SELFNET Self-optimization Use Case [NWC+16] improves the end users perceived QoE in video streaming services. In this case, the sensors are deployed in the virtualized infrastructure. Sensors monitor network state metrics, new Ultrahigh Definition (U-HD) video and energy related metrics. The combination of these low level metrics is combined to address innovative HoN composite metrics. SELFNET is able to detect and respond when QoE levels either have fallen below or are predicted to fall below expected levels. Similarly, the energy efficiency of the system can be analyzed to proactively managing the energy use of resources across the network. Meanwhile, the SELFNET Self-Healing Use Case [SRA+16] is applied to reactively or preventively deal with the detected or predicted network failures. The different metrics collected by the sensors give a global view of the available networking resources and services running on virtualized infrastructure. In this way, the system can be adapted to new load situations through the instantiation of virtual load balancers in strategical points. Additionally, SELFNET is able to use the historical references to predict high-resource demands before they happen and take actions about them.

9.5 Summary

This chapter describes the SELFNET Monitoring and Discovery framework. Similarly, the workflows of the tasks performed by the modules of the framework are defined. Then, the implementation details and the GUI are detailed.

Chapter 10

Conclusions and Future Works

This thesis presents research results from the study of SDN and NFV in terms of Monitoring, Quality of Service and Network Management. Similarly, the research has been motivated and inspired by an analysis practice of 5G requirements, visions and Key Performance Indicators (KPIs).

As stated at the beginning of this research document, the main objective consisted of review the state of the art of SDN, NFV and the challenges of these paradigms to improve the network services. Similarly, the vision of a 5G self-organized ecosystem of networking, computing and storage resources has gained importance for the research community. The need of replace the manual reconfiguration of equipments with self-organized capabilities opens the possibility of a new market based on customized network services. In this context, this Thesis is part of SELFNET Project, which aims to design and implement an autonomic network management framework. SELFNET involves a number of essential management tasks such as automated network monitoring, autonomic network maintenance, automated deployment of network tools and automated network service provisioning.

With the advent of NFV as a necessary complement to SDN for enabling the customization of network services, the use of both technologies for providing self-management capabilities is the basis of SELFNET architecture. SELFNET integrates the self-management paradigm with the use of data mining, learning algorithms, pattern recognition to identify the network behaviour and automatically take decisions to reduce the risk of failure or QoS degradation. In this way, the system is able to provide smart autonomic management of network functions in order to resolve network problems or improve the QoS/QoE. The SELFNET architecture follows a strictly layered approach. The main layers and sublayers have been defined in agreement with the available NFV and SDN proposed by ETSI NVF and Open Networking Foundation respectively but it goes a step beyond by combining both.

Within the main objectives of SELFNET, the monitoring and collection of relevant network information is a key challenge. SELFNET project has the potential to monitor the network status and service performance in order to provide intelligent and advanced capabilities. This research proposes the SELFNET Monitoring and Discovery Framework, which is able to collect data from five different sources. It focuses on gathering and

storing the information (low level metrics) related to physical and virtual devices, cloud environments, flow metrics, SDN traffic and sensor data. This work lay the foundations to provide a contextaware model based on a customizable set of low-level metrics. As a result, the proposed framework will facilitate not only the querying process of gathered information but also the management of large amounts of data from heterogeneous data sources. The information provided will be available for further aggregation, correlation and analysis processes.

Furthermore, the design and specification of the framework is implemented based on well-known tools. The different available monitoring solutions have been investigated and evaluated, and the suitable solutions have been adopted, reused and enhanced in order to fulfill the framework requirements. In particular, LibreNMS, Opendaylight TSDR, OpenStack Telemetry (Ceilometer) and Ceilosca have been selected as the starting point for the prototype implementation.

Following the SELFNET objectives of improve the monitoring functionalities, the present research also developed an SDN monitoring framework that intends to reduce the CPU load and hardware resources whilst maintaining high levels of accuracy. The framework proposes an orchestrator module with an adaptive method of polling information requests based on the controller load and network size. The results using an implementation in an OpenFlow controller, a network emulator and video server demonstrated the feasibility of the framework.

Similarly, regarding the quality of service, a SDN framework to provide QoS for different multimedia services is proposed. The framework uses OpenFlow, Network Virtualization and establishes functional boxes and interfaces to test different routing algorithms. The advantages of the framework are tested with the implementation of a network performance and routing algorithms functions. The experiments with video streaming information show a quality optimization (PSNR, SSIM, MOS) in comparison with the best effort engine.

10.1 Future Works

Considering that SDN, NFV and self-management development are at an early stage; this research highlight to touch on some future works that can be conducted.

Regarding self-management on SDN/NFV networks, the SELFNET project has several challenges that can be analyzed. Once the low level metrics provided by the Monitoring and Discovery are available, the aggregation and correlation procedures are required. In this context, it is necessary the coordination of several aggregation and correlation rules requested by service provider. The definition of relevant high level network metrics is a constant topic of discussion. Similarly, the possibility to use the network related metrics to detect patterns and predict future network status is seen as a research area that can be studied in depth.

There is also the need of a *Situational Awareness* (SA) model, which takes into account the traditional network management and the expected dynamism of 5G environments. Furthermore, it is important to establish and formalize the contextual analysis of the

information provided by the Monitoring and Discovery framework. The analysis model should enable the customization of parameters, analysis functions and diagnosis related rules. In this way, the system is able to diagnosis the network state and predict potential issues. Similarly, the inclusion of diagnosis and prediction capabilities in the context of 5G networks facilitates the decision-making process of mobile services and applications and then facilitates reactive and proactive responses.

Regarding SDN monitoring, the future challenges comprise the improvement of the different monitoring algorithms including new metrics such as connection establishment time, jitter or packet duplication. Similarly, the coordination between data and control planes to avoid overheads in network hardware and controller as well as the coexistence and integration between traditional and SDN networks is an open challenge. In this context, the experimentation with separated hardware elements (e.g. different physical machines for mininet and floodlight) would provide more accurate results.

Regarding QoS optimization in SDN networks, the challenges for future research include the improvement of Network Performance and QoS Routing Algorithms. The possibility of adding additional metrics (e.g. delay or jitter) as part of the cost function can be studied in depth. Furthermore, the development of experiments with high bandwidth links, big topologies, large number of flows and low loss rates in real environments and different NOS controllers is also challenging. Similarly, multipath forwarding and tunneling can optimize the network resources in function of application priority.

Bibliography

[5G 17a] 5G Americas. http://www.5gamericas.org/es/, April 2017.

[5G 17b] 5G Forum. http://www.5gforum.org/, April 2017.

[5GM17] Fifth Generation Mobile Communication Promotion Forum (5GMF). http://5gmf.jp/en, April 2017.

[AAKS98] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network*, 12(3):37–45, May 1998.

[ABC+14] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. Soong, and J. C. Zhang. What will 5G Be? *IEEE Journal on selected areas in communications*, 32(6):1065–1082, June 2014.

[AGHE+15] N. Amit, A. Gordon, N. Har El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir. Bare-metal performance for virtual machines with exitless interrupts. *Communications of the ACM*, 59(1):108–116, 2015.

[AHGZ16] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati. Network Function Virtualization in 5G. *IEEE Communications Magazine*, 54(4):84–91, 2016.

[AIS+14] P. K. Agyapong, M. Iwamura, D. Staehle, W. Kiess, and A. Benjebbour. Design Considerations for a 5G Network Architecture. *IEEE Communications Magazine*, 52(11):65–75, November 2014.

[AMO93] R. K. Ahuja, T. L Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, February 1993.

[BAE+13] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin. An OpenNaaS based SDN Framework for Dynamic QoS control. In *Proceedings of the IEEE SDN for Future Networks and Services*, pages 1–7, Anchorage, Alaska USA, November 2013.

[BASS11] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: a Cloud Networking Platform for Enterprise Applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 8, 2011.

[BAX+16] T. S. Buda, H. Assem, L. Xu, D. Raz, U. Margolin, E. Rosensweig, D. R. Lopez, M.-I. Corici, M. Smirnov, and R. Mullins. Can Machine Learning aid in Delivering New Use Cases and Scenarios in 5G? In *Proceedings of the Network Operations and Management Symposium 2016*, pages 1279–1284, Istanbul, Turkey, April 2016.

[BCAB13] M. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. PolicyCop: an Autonomic QoS Policy Enforcement Framework for Software Defined Networks. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, November 2013.

[BGH+14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O Connor, P. Radoslavov, and W. Snow. ONOS: Towards an Open, Distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[BGMB14] N. Baldo, L. Giupponi, and J. Mangues-Bafalluy. Big Data Empowered Self Organized Networks. In *Proceedings of the 20th European Wireless Conference*, pages 1–8, Barcelona, Spain, May 2014.

[BHL+14] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski. Five Disruptive Technology Directions for 5G. *IEEE Communications Magazine*, 52(2):74–80, February 2014.

[BRL+14] J. Blendin, J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer. Position Paper: Software-defined Network Service Chaining. In *2014 Third European Workshop on Software Defined Networks*, pages 109–114, 2014.

[BTAS14] B. Bangerter, S. Talwar, R. Arefi, and K. Stewart. Networks and Devices for the 5G Era. *IEEE Communications Magazine*, 52(2):90–96, February 2014.

[Cal99] K. Calvert. Architectural Framework for Active Networks Version 1.0, 1999.

[CB10] N.M. Chowdhury and R. Boutaba. A Survey of Network Virtualization. *Computer Networks*, 54(5):862–876, 2010.

[CBAB14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, Krakow,Poland, May 2014.

[cbe17] Cbench. https://github.com/andi-bigswitch/oflops/tree/master/cbench, April 2017.

[CCF+05] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Van der Merwe. Design and Implementation of a Routing Control Platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation*, volume 2, pages 15–28, Berkeley, CA, USA, May 2005.

[CDLL15] L. M. Contreras, P. Doolan, H. Lønsethagen, and D. R. López. Operational, Organizational and Business Challenges for Network Operators in the Context of SDN and NFV. *Computer Networks*, 92:211–217, December 2015.

[cei17a] Ceilometer Metrics. http://docs.openstack.org/admin-guide/telemetry-measurements.html, April 2017.

[cei17b] Ceilosca. https://www.openstack.org/assets/Uploads/Ceilosca-v2-2.pptx, April 2017.

[cei17c] Ceilosca Project. https://www.openstack.org/summit/tokyo-2015/videos/presentation/ceilometer-monascaceilosca, April 2017.

[CFP+07] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 1–12, Stockholm, Sweden, August 2007.

[CFSD90] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[Cla04] B Claise. RFC 3954 - Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[CNP03] A. Courtney, H. Nilsson, and J. Peterson. The Yampa Arcade. In *Proceedings of the ACM Workshop on Haskell*, pages 7–18, New York, NY, USA, August 2003.

[cro17] EU CROWD Project; Connectivity Management for eneRgy Optimised Wireless Dense networks. Project reference: 318115. Funded under: FP7-ICT. http://www.ict-crowd.eu/, April 2017.

[CZA+15] S. Chen, J. Zhao, M. Ai, D. Liu, and Y. Peng. Virtual RATs and a Flexible and Tailored Radio Access Network Evolving to 5G. *IEEE Communications Magazine*, 53(6):52–58, 2015.

[Dev17] Google Developers. Firebase Cloud Messaging. https://firebase.google.com/docs/cloud-messaging/, March 2017.

[DGK+13] P. Demestichas, A. Georgakopoulos, D. Karvounas, K. Tsagkaris, V. Stavroulaki, J. Lu, C. Xiong, and J. Yao. 5G on the Horizon: Key Challenges for the Radio-access Network. *IEEE Vehicular Technology Magazine*, 8(3):47–53, July 2013.

[EBSB11] R Enns, M Bjorklund, J Schoenwaelder, and A Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Historic), June 2011.

[ECT13] H. E. Egilmez, S. Civanlar, and A. M. Tekalp. An Optimization Framework for QoS-enabled Adaptive Video Streaming over OpenFlow Networks. *IEEE Transactions on Multimedia*, 15(3):710–715, April 2013.

[EDBT12] H.E. Egilmez, S.T. Dane, K.T. Bagci, and A.M. Tekalp. OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with End-to-end Quality of Service over Software-Defined Networks. In *2012 Asia-Pacific Signal Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1–8, December 2012.

[EHE15] R. El-Hattachi and J. Erfanian. Next Generation of Mobile Networks. White paper, NGMN Alliance, February 2015.

[Eri13] D. Erickson. The Beacon Openflow Controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, pages 13–18, New York, NY, USA, August 2013.

[ETS] ETSI Industry Specification Group (ISG). Network Function Virtualization (NFV) use cases.

[ETS13a] ETSI Industry Specification Group (ISG). Network Function Virtualization (NFV) Architectural Framework, 2013.

[ETS13b] ETSI Industry Specification Group (ISG). Network Functions Virtualization White Paper, 2013.

[ETS14] ETSI Industry Specification Group (ISG). Management and Orchestration (ETSI MANO), 2014.

[FBMP12] P. Fonseca, R. Bennesby, E. Mota, and E. Passito. A Replication Component for Resilient OpenFlow-based Networking. In *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pages 933–939, Maui, HI, USA, April 2012.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol–HTTP/1.1. RFC 2616: Hypertext Transfer Protocol–HTTP/1.1, June 1999.

[FGR+13] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for Software Defined Networks. *IEEE Communications Magazine*, 51(2):128–134, February 2013.

[FHF+11] N. Foster, R. Harrison, M. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A Network Programming Language. In *Proceedings of the 16th ACM international Conference on Functional Programming*, volume 46, pages 279–291, New York, NY, USA, September 2011.

[flo17] Floodlight. http://www.projectfloodlight.org/, April 2017.

[FP99] M. Flatin and J. Philippe. Push vs. Pull in Web-based Network Management. In *In Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 3–18, May 1999.

[FRZ14] N. Feamster, J. Rexford, and E. Zegura. The Road to SDN: An Intellectual History of Programmable Networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[GAA+13] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments. *Computer Networks*, pages 122–136, April 2013.

[GEB+13] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, pages 15–20, Hong Kong, China, August 2013.

[GHM+05] A. Greenberg, G. Hjaimtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, October 2005.

[GJ15] A. Gupta and R. K. Jha. A Survey of 5G Network: Architecture and Emerging Technologies. *IEEE access*, 3:1206–1232, July 2015.

[GKM+16] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Xilouris, C. Sakkas, and A. Kourtis. An Integrating Framework for Efficient NFV Monitoring. In *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pages 1–5, 2016.

[GKP+08a] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[GKP+08b] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[GMUJ16] J. Garay, J. Matias, J. Unzilla, and E. Jacob. Service Description in the NFV Revolution: Trends, Challenges and a Way Forward. *IEEE Communications Magazine*, 54(3):68–74, 2016.

[gno17] Gnocchi. https://wiki.openstack.org/wiki/Gnocchi, April 2017.

[GRF13] A. Guha, M. Reitblatt, and N. Foster. Machine verified Network Controllers. In *ACM SIGPLAN NOTICES*, volume 48, pages 483–494, June 2013.

[GSB13] M. Gupta, J. Sommers, and P. Barford. Fast, Accurate Simulation for SDN Prototyping. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, pages 31–36, August 2013.

[GVVC15] L. J. García Villalba and A. L. Valdivieso Caraguay. Use Cases Definition and Requirements of the System and its Components. *SELFNET Project*, 2015.

[GVVC16] L. J. García Villalba and A. L. Valdivieso Caraguay. Report and Prototypical Implementation of the Monitoring and Discovery Module. *SELFNET Project*, 2016.

[HHB14] F. Hu, Q. Hao, and K. Bao. A Survey on Software-defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys & Tutorials*, 16(4):2181–2206, May 2014.

[HNS+09] J.-C. Hourcade, Y. Neuvo, R. Saracco, W. Wahlster, and R. Posch. Future Internet 2020: Visions of an Industry Expert Group. Panel report, May 2009.

[HP15] J. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. Technical report, 2015.

[HSMA14] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal. NFV: State of the Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC). *IEEE Network*, 28(6):18–26, 2014.

[IMT17] IMT-2020 (5G) Promotion Group. http://www.imt-2020.cn/en/introduction, April 2017.

[itu96] ITU-T P.800. Methods for Subjective Determination of Transmission Quality - Series P: Telephone Transmission Quality; Methods for Objective and Subjective Assessment of Quality. pages 1–37, August 1996.

[JKM+13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu. B4: Experience with a Globally-deployed Software Defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, August 2013.

[JP13] R. Jain and S Paul. Network Virtualization and Software Defined Networking for Cloud Computing: a Survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[JSMR01] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó. Lagrange Relaxation Based Method for the QoS Routing Problem. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 859–868, Anchorage, Alaska USA, April 2001.

[KF13] H. Kim and N. Feamster. Improving Network Management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114–119, February 2013.

[KFG15] D. King, A. Farrel, and N. Georgalas. The Role of SDN and NFV for Flexible Optical Networks: Current Status, Challenges and Opportunities. In *17th International Conference on Transparent Optical Networks (ICTON)*, pages 1–6, 2015.

[KGH13] M. Karl, J. Gruen, and T. Herfet. Multimedia Optimized Routing in OpenFlow Networks. In *Proceedings of the 19th IEEE International Conference on Networks*, pages 1–6, Singapore, Singapore, December 2013.

[KK13] Shashi Kiran and Gary Kinghorn. Cisco Open Network Environment: Bring the Network Closer to Applications, 2013.

[KRV+15] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, December 2015.

[KRW03] J. Klaue, B. Rathke, and A. Wolisz. Evalvid-A Framework for Video Transmission and Quality Evaluation. In *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 255–272. 2003.

[KSC+11] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards. Communicating with Caps: Managing Usage Caps in Home Networks. In *Proceedings of the ACM SIGCOMM Conference*, pages 470–471, New York, NY, USA, August 2011.

[KSKD+12] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely. Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking. In *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks*, volume 1, pages 1–5, Split, Croatia, September 2012.

[KSL+10] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula. Automated and Scalable Qos Control for Network Convergence. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, volume 10, pages 1–1, Berkeley, CA, USA, 2010.

[KT217] Open Platform for NFV (OPNFV). https://www.opnfv.org/, April 2017.

[kt317a] EU CONTENT Project; Convergence of Wireless Optical Network and iT rEsources iN SupporT of Cloud Services. Project reference: 318514 . Funded under: FP7-ICT. http://cordis.europa.eu/project/rcn/106689_en.html, April 2017.

[kt317b] EU METIS-II Project; Mobile and Wireless Communications Enablers for Twenty-Twenty (2020) Information Society-II. Project reference: 671680 . Funded under: H2020-ICT-2014-2. https://5g-ppp.eu/metis-ii/, April 2017.

[KT417a] EU CHARISMA Project; Converged Heterogeneous Advanced 5G Cloud-RAN Architecture for Intelligent and Secure Media Access. Project Reference: 671704. Funded under: H2020-ICT-2014-2. http://www.charisma5g.eu/, April 2017.

[KT417b] EU Flex5Gware Project. Flexible and Efficient Hardware/Software Platforms for 5G Network Elements and Devices. Project Reference: 671563. Funded under: H2020-ICT-2014-2. http://www.flex5gware.eu/, April 2017.

[KT417c] EU SONATA Project. Service Programing and Orchestration for Virtualized Software Networks. Project Reference: 671517. Funded under: H2020-ICT-2014-2. http://www.sonata-nfv.eu/, April 2017.

[LCMP12] P. Le Callet, S. Möller, and A Perkis. Qualinet White Paper on Definitions of Quality of Experience. *European Network on Quality of Experience in Multimedia Systems and Services*, 3, March 2012.

[LDC13] N. Li, Y. Du, and G. Chen. Survey of Cloud Messaging Push Notification Service. In *In International Conference on Information Science and Cloud Computing Companion (ISCC-C)*, pages 273–279, December 2013.

[LHM10] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, New York, NY, USA, October 2010.

[lib17] LibreNMS. http://www.librenms.org/, April 2017.

[LNR+04] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo. The SoftRouter Architecture. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networking*, pages 1–6, New York, NY, USA, November 2004.

[LTRW16] G. Liu, M. Trotter, Y. Ren, and T. Wood. NetAlytics: Cloud-Scale Application Performance Monitoring with SDN and NFV. In *Proceedings of the 17th International Middleware Conference*, page 8, 2016.

[MAB+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

[mcn17] EU MCN Project; Mobile Cloud Networking. Project reference: 318109. Funded under: FP7. http://www.mobile-cloud-networking.eu/site/, April 2017.

[MFHW12] C. Monsanto, N. Foster, R. Harrison, and D. Walker. A Compiler and Run-time System for Network Programming Languages. In *ACM SIGPLAN NOTICES*, volume 47, pages 217–230, January 2012.

[MGB+09] L. Meyerovich, A. Guha, J. Baskin, G. Cooper, M. Greenberg, and A. Bromfield. Flapjax: A Programming Language for Ajax Applications. In *Proceedings of the 24th ACM conference on Object Oriented Programming Systems Languages and Applications*, volume 44, pages 1–20, New York, NY, USA, October 2009.

[min17] Mininet. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#limits, April 2017.

[Moh15] W. Mohr. The 5G Infrastructure Public-Private Partnership. In *Presentation in ITU GSC-19 Meeting*, 2015.

[mon17] Monasca Project. https://wiki.openstack.org/wiki/Monasca, April 2017.

[MSG+16] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-art and Research Challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, September 2016.

[MVTG14] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a Model-driven Sdn Controller Architecture. In *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[Mye99] A. C. Myers. JFlow: Practical Mostly-static Information Flow Control. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, Texas, USA, January 1999.

[NCC10] E. Ng, Z. Cai, and A.L. Cox. Maestro: A System for Scalable OpenFlow Control. Technical Report, Rice University, December 2010.

[NCC+16] P. Neves, R. Calé, M. R. Costa, C. Parada, B. Parreira, J. Alcaraz-Calero, Q. Wang, J. Nightingale, E. Chirivella-Perez, and W. Jiang. The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm. *International Journal of Distributed Sensor Networks*, 2016:2, 2016.

[NEC15] NEC Corporation. Network Evolution Toward 2020 and Beyond. http://www.nec.com/en/global/solutions/nsp/5g_vision/doc/2020_network.pdf, 2015.

[Net14] E. NetWorld2020. 5G: Challenges, Research Priorities, and Recommendations. *Joint White Paper September*, 2014.

[Nok14] Nokia. 5G Use Cases and Requirements White Paper, July 2014.

[nor17] EU 5G-NORMA Project; 5G Novel Radio Multiservice Adaptive Network Architecture. Project reference: 671584 . Funded under: H2020-ICT-2014-2. https://5gnorma.5g-ppp.eu/, April 2017.

[NRFC09] A. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic Access Control for Enterprise Networks. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, pages 11–18, New York, NY, USA, August 2009.

[NVCGV17] P. Neves, A. L. Valdivieso Caraguay, and L. J. García Villalba. Future Mode of Operations for 5G - The SELFNET Approach Enabled by SDN/NFV. *Computer Standards & Interfaces*, 2017.

[NWC+16] J. Nightingale, Q. Wang, J. M. Calero, , and et. all. QoE-driven, Energy-aware Video Adaptation in 5G Networks: The SELFNET Self-optimisation Use Case. *International Journal of Distributed Sensor Networks*, 2016(12):1–15, 2016.

[OBB+14] A. Osseiran, F. Boccardi, V. Braun, K. Kusume, P. Marsch, M. Maternia, O. Queseth, M. Schellmann, H. Schotten, and H. Taoka. Scenarios for 5G Mobile and Wireless Communications: The Vision of the METIS Project. *IEEE Communications Magazine*, 52(5):26–35, 2014.

[ONF17] ONF. Open Networking Foundation. https://www.opennetworking.org/, April 2017.

[ope09] OpenFlow Switch Specification v.1.0.0, December 2009.

[ope13] OpenFlow Management and Configuration Protocol (OF-Config) v.1.1.1, March 2013.

[ope17a] OpenBaton. http://openbaton.github.io/, April 2017.

[ope17b] OpenStack. https://www.openstack.org/, April 2017.

[PJ12] S. Paul and R. Jain. Openadn: Mobile Apps on Global Clouds using Openflow and Software Defined Networking. In *2012 IEEE Globecom Workshops*, pages 719–723, 2012.

[PL04] P. Phaal and M. Lavine. Sflow version 5, July 2004.

[Pla17] PlanetLab. https://www.planet-lab.org/, April 2017.

[PLHea11] B. Pfaff, B. Lantz, B Heller, and et. all. OpenFlow Switch Specification v.1.1.0. Specification, Open Networking Foundation, February 2011.

[PNC+14] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, and A. Neal. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) Industry Initiative*, September 2014.

[PPA+09] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker. Extending Networking into the Virtualization Layer. In *Proceedings of the ACM SIGCOMM HotNets*, pages 1–6, New York, NY, USA, October 2009.

[PWH13] K. Pentikousis, Y. Wang, and W. Hu. MobileFlow: Toward Software- Defined Mobile Networks. In *IEEE Communications Magazine*, volume 51, pages 44–53, July 2013.

[RFR+12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for Network Update. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 42, pages 323–334, New York, NY, USA, October 2012.

[RMTF09] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster. Securing Enterprise Networks Using Traffic Tainting, August 2009.

[RSC14] D. Raumer, L. Schwaighofer, and G. Carle. MonSamp: A Distributed SDN Application for QoS Monitoring. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 1–9, Krakow,Poland, May 2014.

[SA04] E. Saint-Andrew. RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence, October 2004.

[SAE12] O. Sefraoui, M. Aissaoui, and M. Eleuldj. OpenStack: Toward an Open-source Solution for Cloud Computing. *International Journal of Computer Applications*, 55(3), 2012.

[SBJN16] S. A. R. Shah, S. Bae, A. Jaikar, and S.-Y. Noh. An Adaptive Load Monitoring Solution for Logically Centralized SDN Controller. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–6, October 2016.

[SC15] D. Samsung Center. Samsung 5G VisionElectronics Co. 2015.

[sel17] EU SELFNET Project - Self-Organized Network Management in Virtualized and Software Defined Networks. Project reference: H2020-ICT-2014-2/671672. Funded under: H2020. http://www.selfnet-5g.eu, April 2017.

[SGY+09] R. Sherwood, G. Gibb, K.K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A Network Virtualization Layer. Technical Report, OpenFlow Switch Consortium, October 2009.

[SNC+14] B. Sonkoly, F. Németh, L. Csikor, L. Gulyás, and A. Gulyás. SDN Based Testbeds for Evaluating and Promoting Multipath TCP. In *Proceedings of the IEEE International Conference on Communications 2014*, pages 3044–3050, Sydney, NSW, Australia, August 2014.

[SOP+16] B. Siniarski, C. Olariu, P. Perry, T. Parsons, and J. Murphy. Real-time Monitoring of SDN Networks using Non-invasive Cloud-based Logging Platforms. In *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6, September 2016.

[SRA+16] J. P. Santos, A. Rui, L. Andrade, A. L. Valdivieso Caraguay, L. I. Barona López, and L. J. García Villalba. SELFNET Framework Self-healing Capabilities for 5G Mobile Networks. *Transactions on Emerging Telecommunications Technologies*, 27(9):1225–1232, 2016.

[SSC+12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. A Demonstration of Fast Failure Recovery in Software Defined Networking. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 44, pages 411–414. Thessanoliki, Greece, June 2012.

[SSHC+13] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are We Ready for SDN? Implementation Challenges for Software-Defined Networks. *IEEE Communications Magazine*, 51(7):36–43, July 2013.

[STH14] M. Shibuya, A. Tachibana, and T. Hasegawa. Efficient Performance Diagnosis in OpenFlow Networks Based on Active Measurements. In *Proceedings of the Thirteenth International Conference on Networks*, pages 268–273, Nice, France, February 2014.

[SWL+16] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang. Traffic Engineering in Software-defined Networking: Measurement and Management. *IEEE Access*, 4:3246–3256, 2016.

[tel17a] Telemetry. http://docs.openstack.org/admin-guide/telemetry.html, April 2017.

[tel17b] Telemetry Project. https://wiki.openstack.org/wiki/Telemetry, April 2017.

[TFF+13] Patricia Thaler, Norman Finn, Don Fedyk, Glenn Parsons, and Eric Gray. Ieee 802.1 q, 2013.

[TGG10] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Proceedings of the International Conference on Passive and Active Network Measurement*, pages 201–210, Krakow,Poland, January 2010.

[TGG+12] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On Controller Performance in Software-defined Networks. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, pages 1–6, New York, NY, USA, April 2012.

[tno17] EU T-NOVA Project; Network Functions as-a-Service over Virtualised Infrastructures. Project reference: 619520. Funded under: FP7. http://www.t-nova.eu/, April 2017.

[tsd17] OpenDaylight TSDR. https://wiki.opendaylight.org/view/Project_Proposals:Time_Series_Data_Repository, April 2017.

[uni17] EU UNIFY Project; Unifying Cloud and Carrier Networks. Project reference: 619609. Funded under: FP7. https://www.fp7-unify.eu/, April 2017.

[vADK14] N. L. van Adrichem, C. Doerr, and F. A. Kuipers. OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks. In *Proceedings of the Network Operations and Management Symposium*, pages 1–8, Krakow,Poland, May 2014.

[VCBLBPGV14] A. L. Valdivieso Caraguay, L. I. Barona López, A. Benito Peral, and L. J. García Villalba. SDN: Evolution and Opportunities in the Development IoT Applications. *International Journal of Distributed Sensor Networks*, 2014, May 2014.

[VCBLGV13] A. L. Valdivieso Caraguay, L. I. Barona López, and L. J. García Villalba. Evolution and Challenges of Software Defined Networking. In *Proceedings of 2013 Workshop on Software Defined Networks for Future Networks and Services*, pages 61–67, November 2013.

[VCBLGV15] A. L. Valdivieso Caraguay, L. I. Barona López, and L. J. García Villalba. An Overview of Integration of Mobile Infrastructure with SDN/NFV Networks. pages 259–265, 2015.

[VCGV] A. L. Valdivieso Caraguay and L. J. García Villalba. Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks. *Sensors (Submitted December 2016).*

[VCPGV] A. L. Valdivieso Caraguay, J. A. Puente, and L. J. García Villalba. An Optimization Framework for Monitoring of SDN/OpenFlow Networks. *International Journal of Ad Hoc and Ubiquitous Computing (Accepted September 2015).*

[VCPGV15] A. L. Valdivieso Caraguay, J. A. Puente, and L. J. García Villalba. Framework for Optimized Multimedia Routing over Software Defined Networks. *Computer Networks*, 92:369–379, 2015.

[vid17] Highway. http://www2.tkn.tu-berlin.de/research/evalvid/qcif.html, April 2017.

[VKF12] A. Voellmy, H. Kim, and N. Feamster. Procera: A Language for High- Level Reactive Network Control. In *Proceedings of the First Workshop on Hot topics in Software Defined Networks*, pages 43–48, Helsinki, Finland, August 2012.

[VMK04] P. Van Mieghem and F. A. Kuipers. Concepts of Exact QoS Routing Algorithms. *IEEE/ACM Transactions on Networking*, 12(5):851–864, October 2004.

[vS13] K. van Surksum. Paper: VMware NSX Network Virtualization Design Guide. 2013.

[VW12] A. Voellmy and J. Wang. Scalable Software Defined Network Controllers. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 42, pages 289–290, New York, NY, USA, August 2012.

[WT01] T. Wolf and J. Turner. Design Issues for High-performance Active Routers. *IEEE Journal on Selected Areas in Communications*, 19(3):404–409, March 2001.

[XAY+16] L. Xu, H. Assem, I. G. B. Yahia, T. S. Buda, A. Martin, D. Gallico, M. Biancani, A. Pastor, P. Aranda, and M. Smirnov. CogNet: A Network Management Architecture Featuring Cognitive Capabilities. In *Proccedings of the European Conference on Networks and Communications*, pages 325–329, 2016.

[YCY+15] Y.-Y. Yang, W.-H. Cheng, C.-T. Yang, S.-T. Chen, and F.-C. Jian. The Implementation of Real-Time Network Traffic Monitoring Service with Network Functions Virtualization. In *2015 International Conference on Cloud Computing and Big Data (CCBD)*, pages 279–286, 2015.

[YDAG04] L. Yang, R. Dantu, T. Anderson, and R. Gopal. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746 (Informational), April 2004.

[ZCB10] Q. Zhang, L. Cheng, and R. Boutaba. Cloud Computing: State-of-the-art and Research Challenges. *Journal of internet services and applications*, 1(1):7–18, April 2010.

[zet16] The Zettabyte EraTrends and Analysis, June 2016.

Part III

Descripción de la Investigación

Capítulo 11

Introducción

Las arquitecturas de red tradicionales experimentan considerables limitaciones en el provisionamiento de nuevos servicios de red, especialmente en términos de operatividad, seguridad, calidad de servicio, costos de operación y mantenimiento. La íntima unión entre el plano de datos y plano de control en los dispositivos de red, el creciente número protocolos y el acceso privativo y limitado al código fuente son las principales restricciones que impiden la innovación y personalización de servicios. Las actuales estrategias de mitigación de los problemas de red consisten principalmente en la reconfiguración manual de los equipos, los cuales causan interrupciones en la operación normal de la red. En algunos casos, la única solución factible incluye la instalación de equipamiento y funcionalidades adicionales, tales como encaminadores, balanceadores, firewalls, DPIs, entre otros.

La creación y personalización de nuevos servicios requieren un entorno abierto, en donde los desarrolladores tengan libertad y capacidad de modificar el comportamiento de la red sin afectar a otros usuarios. En este sentido, una solución factible es seguir los alcances logrados en sistemas computacionales, en donde los desarrolladores pueden crear sus propias aplicaciones usando un lenguaje de programación de alto nivel. Los programas pueden ejecutarse en diferentes equipos gracias a la abstracción de los recursos físicos provistos por el sistema operativo. En este contexto, las Redes Definidas por Software SDN y la Virtualización de las Funciones de Red NFV surgen como soluciones apropiadas para obtener los objetivos propuestos.

SDN propone la separación de los planos de datos y control en los dispositivos de red, habilitando su independiente evolución y desarrollo. De igual manera, se cambia la visión individual del equipo hacia una visión global de toda la red. Por su parte, NFV promueve la migración de los equipos tradicionales de red que cumplen una función específica (DPI, firewall, balanceadores) por aplicaciones de software o funciones de red NF, los cuales pueden ser instanciados en una infraestructura virtualizada. Ambas arquitecturas son complementarias y su integración podrían potencialmente brindar un entorno de red abierto para desarrolladores y administradores de la red. Sin embargo, su desarrollo se encuentra en etapa inicial y múltiples retos han sido identificados antes de lograr una completa madurez tecnológica.

De igual manera, dentro de la visión de la futuras redes móviles y servicios 5G, la

disminución en los gastos de gestión y mantenimiento a travez de la automatización de operaciones es uno de los principales retos. El creciente número de dispositivos móviles conectados al internet, tales como teléfonos inteligentes o tabletas no pueden ser gestionados por medio de la configuración manual de la infraestructura de red, por ejemplo cuando se requiere mitigar un problema o amenaza (fallos de enlaces, caída de servicio, ataques de seguridad, degradación de la calidad de servicio, entre otros). En este sentido, el presente trabajo de investigación centra sus esfuerzos en la integración de SDN, NFV, inteligencia artificial, calidad de servicio, con el objetivo de brindar una solución de gestión autónoma escalable, extensible y abierta para redes móviles 5G.

Adicionalmente, el presente trabajo se enfoca en las tareas de monitorización y descubrimiento. Dichas operaciones son de vital importancia en la gestión de redes, debido a que todas las decisiones que tomen los agentes encargados del mantenimiento y operación dependen de la rapidez y precisión de la información enviada por los sistemas de monitorización. En este contexto, es necesario una arquitectura de monitorización distribuida para infraestructura de red heterogéneos, el cual permita a los desarrolladores tener un adecuado conocimiento del estado de la red.

Las siguientes secciones resumen la problemática de la investigación, cuales son las principales motivaciones y los objetivos del estudio. Así mismo, se presenta un corto resumen de las principales contribuciones del presente trabajo y su estructura.

11.1 Problema de Investigación

Con el objetivo de clarificar el problema de investigación, la presente sección brinda las bases para la discusión de los temas investigados, los mismos que serán ampliamente explicados en los siguientes apartados.

Durante la etapa de búsqueda de información en la literatura actual, se constató que el desarrollo de valores añadidos y personalización de los servicios de red se encuentra limitado por la rigidez de las arquitecturas tradicionales. Las arquitecturas actuales se encuentran basadas en un número fijo de nodos, cada uno con su dirección IP estática, conectada a una interfaz física de red en un único dominio. Sin embargo, esta composición ha evolucionado hacia millones de dispositivos móviles conectados a diferentes proveedores de servicios y solicitando baja latencia y servicios en tiempo real. Esta evolución también exige cambios importantes en la manera de gestionar y brindar mantenimiento a la infraestructura de red. Actualmente, las operadoras invierten grandes recursos para detectar y mitigar manualmente las incidencias en la red. En este sentido, las nuevas soluciones de operación y mantenimiento deben incluir servicios de gestión autónomos. Con este propósito, los principios de diseño de arquitecturas 5G dan a conocer la urgente necesidad de adoptar soluciones autónomas con el fin de reducir gastos de operación y mantenimiento (Capex, Opex).

En este contexto, el presente trabajo de investigación es parte del proyecto europeo Self-Organized Network Management in Virtualized and Software Defined Networks H2020 SELFNET. SELFNET tiene como objetivo el diseño e implementación de una arquitectura de gestión autónoma para redes 5G. SELFNET utiliza los principios de SDN y NFV como

línea base para brindar capacidades de gestión autónomas para redes heterogéneas que permitan prevenir o mitigar los principales problemas en la red. Con este objetivo, el presente proyecto de investigación incluye el estudio de 5G, SDN, NFV, computación en la nube, junto con algoritmos innovadores que permitan brindar "inteligencia" en escenarios complejos de red.

Dentro de las funcionalidades claves brindadas por SELFNET, se incluye el diseño de una arquitectura de monitorización que permita la recolección de información, no sólo de métricas tradicionales de bajo nivel, sino también la inclusión de métricas de alto nivel ajustables a casos de uso particulares. En este sentido, uno de los propósitos del presente trabajo incluye el despliegue automático de aplicaciones SDN/NFV en la infraestructura virtualizada que faciliten el monitoreo de métricas de sensores distribuidos a lo largo de toda la red.

11.2 Motivación

En general, el surgimiento de los principios de SDN y NFV han ganado la atención de la comunidad ciéntifica. La posibilidad de superar la rigidez y complejidad que presentan las arquitecturas tradicionales han originado multiples oportunidades a los desarrolladores y proveedores de servicios. SDN y NFV han cambiado la visión de una infraestructura de red monolítica que requiere altos costos de mantenimiento y procedimientos complejos de actualización hacia una plataforma abierta de servicios integrada, flexible e interconectada con diferentes operadoras. Estas nuevas formas de visualizar las redes han creado nuevas oportunidades de mercado basados en servicios de valor añadido al igual que nuevos modelos de negocio con amplios márgenes de ganancia.

De igual manera, se ha reconocido ampliamente que el nuevo ecosistema de infraestructura móvil 5G estará basada en los principios propuestos por SDN y NFV. Estas tecnologías serán la base fudamental para brindar un entorno flexible y abierto a nuevos servicios. De este modo, múltiples funciones virtuales diseñados con un propósito específico podrán ser instanciados en diferentes dominios independientes que comparten una misma infraestructura física. En este contexto, el presente trabajo de investigación tiene como motivación la introducción de capacidades de gestión autónomas para ambientes de red virtualizados. De este modo, las diferentes funciones virtuales podrán ser gestionadas e instanciadas de manera automática con el fin de resolver o mitigar problemas de red, o en su caso prevenir potenciales amenazas, sin causar degradación de la calidad de servicio de los usuarios. Sin embargo, la materialización de estos objetivos trae consigo múltiples retos.

Además, el presente trabajo centra su atención en uno de los aspectos fundamentales de los sistemas de gestión: la monitorización y descubrimiento. La presente investigación propone una arquitectura que permite la monitorización y descubrimiento de sistemas basados en arquitecturas SDN/NFV. Es evidente que el éxito (o fracaso) de las tareas de gestión autónomas proactivas y reactivas dependen de la efectividad y precisión del sistema en tener conocimiento del estado actual de la red. De igual manera, la inclusión de métricas del estado de la red de alto nivel HoN permite a los operadores reducir de manera

considerable la cantidad de información recopilada. En este contexto, el presente trabajo presenta una arquitectura de monitorización que facilita la recogida de información de diferentes fuentes de información basados en SDN/NFV a lo largo de la infraestructura de la red.

11.3 Objetivos

La presente investigación toma como base el estado del arte de los temas 5G, SDN, NFV y tiene como objetivo principal el brindar capacidades de gestión autónomas con el fin de reducir los gastos de gestión y mantenimiento en arquitecturas móviles 5G. Considerando los retos del objetivo principal, se han presentado tres objetivos específicos. El primer objetivo es optimizar el proceso de monitorización en redes SDN. Por su parte, el segundo objetivo es la mejora de la calidad de servicio para redes SDN. El tercer objetivo es el diseño e implementación de una arquitectura de monitorización y descubrimiento para redes SDN/NFV.

Con el fin de lograr el objetivo principal y los objetivos específicos mencionados anteriormente, las siguientes actividades se han llevado a cabo durante la investigación:

1. Revisión del estado del arte en relación a las tecnologías 5G, SDN y NFV.
2. Diseño y descripción de la arquitectura SELFNET de gestión autónoma para redes SDN/NFV.
3. Diseño, implementación y pruebas de una arquitectura de optimización del proceso de monitorización de redes SDN.
4. Diseño, implementación y pruebas de una arquitectura de optimización de la calidad de servicio para redes SDN.
5. Diseño, descripción e implementción de la arquitectura de monitorización y descubrimiento en base a los principios de SELFNET.

11.4 Resumen de las Contribuciones

Los resultados de la presente tesis son organizados en diferentes áreas de conocimiento. Los mismos incluyen los siguientes áreas: Redes Definidas por Software SDN, Virtualización de las Funciones de Red NFV, Redes Móviles 5G, Gestión de Redes y Calidad de Servicio - Calidad de Experiencia (QoS/QoE). La Figura 11.1 resume las contribuciones y las respectivas áreas de conocimiento.

En este sentido, las contribuciones enfocadas en SDN son presentados en [VCBLGV13] y [VCBLBPGV14]. Así mismo, las contribuciones publicadas en [VCPGV15] y [VCBLGV15] también incluyen las áreas de calidad de servicio y calidad de experiencia (QoS/QoE). La propuesta publicada en [VCPGV] es la primera contribución que integra áreas de SDN, NFV con las bases de redes móviles 5G. El resto de las contribuciones publicadas en [SRA+16], [NCC+16], [VCGV], [GVVC15], [GVVC16] incluyen además

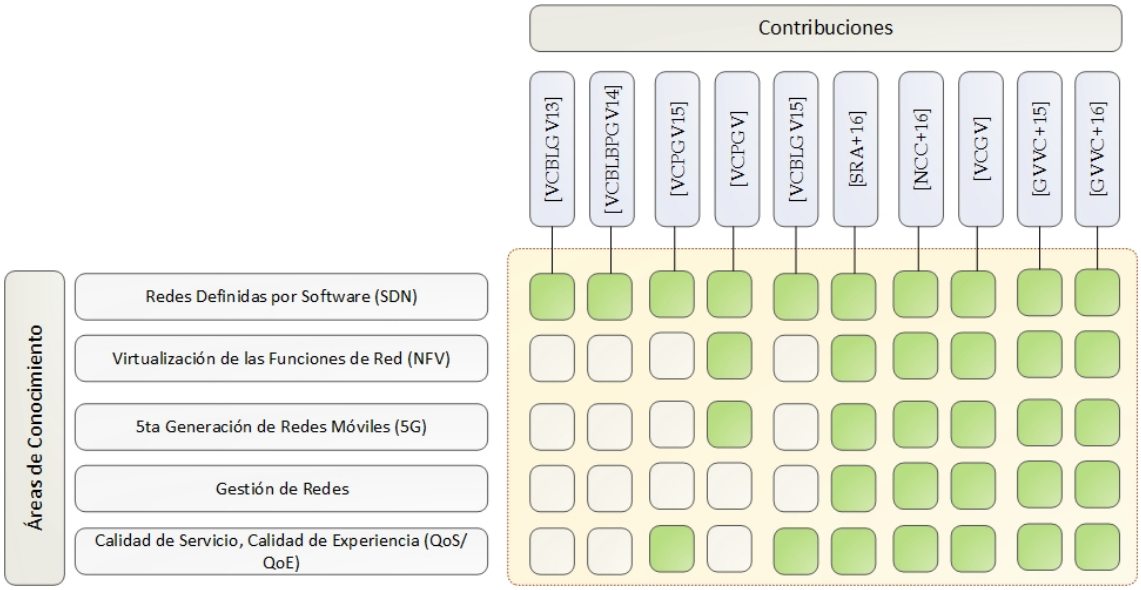


Figura 11.1: Contribuciones de la tesis

temas de gestión de red. En otras palabras, dichas contribuciones incluyen todas las áreas analizadas en el presente trabajo de investigación.

11.5 Estructura del Trabajo

El presente trabajo se estructura como sigue:

El Capítulo 12 presenta la arquitectura SELFNET. Este capítulo resalta las ventajas de SELFNET como solución óptima para brindar servicios de gestión autonómica en redes SDN/NFV. En el capítulo se describe la arquitectura de alto nivel, así como las capas y subcapas que la conforman.

El Capítulo 13 presenta la arquitectura de monitorización y descubrimiento para redes SDN/NFV. En el capítulo se describe la arquitectura de alto nivel y el contexto operacional. Así mismo, se exponen los diferentes elementos, capas y subcapas de la arquitectura. Finalmente, se presentan los detalles de la implementación de la arquitectura.

El Capítulo 14 expone las principales conclusiones de este trabajo, así como algunas posibles líneas futuras de investigación.

11.6 Audiencia de la Tesis

Los requisitos previos para comprender adecuadamente las contribuciones de la presente tesis no son elevados. Con el fin de presentar un documento autocontenido, se han repetido definiciones y conceptos disponibles en la literatura. Es posible que el lector necesite un nivel básico de conocimientos en protocolos de red, gestión de red e infraestructuras móviles. La bibliografía adjunta permitirá al lector encontrar información adicional con respecto a las líneas de investigación presentadas en el presente documento.

Capítulo 12

SELFNET Gestión Autónoma en Redes SDN/NFV

Este capítulo hace una revisión de los principales avances en redes autogestionadas basadas en SDN/NFV. Asimismo, el proyecto SELFNET y sus componentes son descritos en los próximos apartados. Este capítulo está organizado en 10 secciones. La sección 12.1 hace una introducción al capítulo. La sección 12.2 hace una revisión de la gestión de redes con SDNNFV. En la sección 12.3 presenta el proyecto SELFNET. Las secciones siguientes describen cada una de las capas y subcapas del framework SELFNET. La sección 12.4 describe la Capa de Infraestructura. En la sección 12.5 se describe la Capa de Datos de Red, mientras que la sección 12.6 presenta la Capa de Control SON. A su vez, la sección 12.7 describe la Capa Autónoma SON. Por otra parte, la sección 12.8 presenta la Capa de Gestión y Orquestación NFV y en la sección 12.9 se describe la Capa de Acceso SON. Finalmente, la sección 12.10 presenta el resumen de este capítulo.

12.1 Introducción

La gestión y personalización de servicios de red se ha visto limitada por la rigidez de las arquitecturas de red tradicionales y por el incremento tanto de los costes de capital como operacionales. En la actualidad, la solución a problemas de red comunes, como fallos en los enlaces, ataques de seguridad, degradación de la QoS o de la QoE, congestión, entre otros, requieren la intervención directa de los operadores de red. La reconfiguración manual de dispositivos o la instalación de nuevos equipos (encaminadores, servidores NAT, cortafuegos) compromenten la operatividad de la red y repercuten negativamente sobre los niveles de servicio acordados en los SLAs. De forma similar, la creación de servicios innovadores de valor añadido se ve limitada por el hardware y software propietarios y que, en algunos casos, deben únicamente pertenecer al mismo proveedor. Esas limitaciones hacen que las arquitecturas de redes tradicionales se muestren inviables para satisfacer las necesidades actuales de los usuarios, las empresas y los operadores de red.

La solución propuesta para hacer frente a los desafíos actuales ha sido conducida por los avances logrados por la ingeniería de software, en la que los desarrolladores pueden crear sus propias aplicaciones usando lenguajes de programación de alto nivel. Estos programas

pueden ser ejecutados en diversos equipos debido a la abstracción de recursos que los OS son capaces de brindar. En este contexto, SDN y NFV se muestran como potenciales alternativas para lidiar con los recientes desafíos. SDN propone el desacoplamiento de los planos de control y datos en los dispositivos de red, posibilitando su desarrollo y evolución independiente, además de una visión centralizada de la red. NFV promueve la migración de funcionalidades de red típicamente desplegadas en dispositivos (DPI, cortafuegos, balanceadores de carga) a paquetes de software o funciones de red NF que puedan ser instanciadas en una infraestructura virtualizada. Ambas arquitecturas son complementarias y potencialmente integrables para ofrecer a los desarrolladores un entorno de red abierto. Este capítulo describe SDN, NFV, y su evolución en los últimos años. Asimismo analiza la gestión de redes, sus oportunidades y desafíos en el futuro.

Adicionalmente, tanto el crecimiento exponencial de los dispositivos móviles como el advenimiento de la computación en la nube trajeron consigo desafíos adicionales a los operadores de red y proveedores de servicio. Se requiere también, un decremento radical de operaciones de gestión de red integradas que no afecten negativamente la calidad de servicio QoS/QoE ni la seguridad. De forma similar, se promueve un nuevo modelo que integre el acceso y la gestión de recursos móviles. Se espera que las futuras redes 5G proporcionen no sólo una mejor ancho de banda, sino también un modelo de control heterogéneo, simplificado y unificado. Los costes de gestión deben ser reducidos a través de la automatización de operaciones. En este contexto, se presenta como desafío fundamental la reducción de costos operacionales por medio del desarrollo de arquitecturas de gestión escalables que incluyan técnicas de minería de datos, reconocimiento de patrones, algoritmos de aprendizaje automático, etc. Este capítulo describe también el estado del arte y los recientes avances en este campo.

El proyecto SELFNET [sel17] hace uso de los principios de SDN y NFV para proporcionar una arquitectura de gestión autónoma de funciones de red. De este modo, se facilita la resolución de problemas de red y se mejora la calidad de servicio QoS percibida por los usarios. La auto-gestión es facilitada por medio del uso minería de datos, algoritmos de aprendizaje automático, reconocimiento de patrones, etc. acoplados a entornos móviles 5G. Asimismo, el sistema es capaz de decidir las mejores acciones que mitiguen de forma automática problemas de red. La arquitectura de SELFNET está compuesta por diversas capas: Infraestructura, Red Virtualizada, Control SON, Acceso a la Red y SON Autónoma. Dentro de la capa SON Autónoma, la subcapa de Monitorización y Análisis es una de las que presenta importantes desafíos. Dicha subcapa, a su vez, está dividida en tres módulos: Monitorización y Descubrimiento, Aggregación y Correlación, y Análisis.

12.2 Gestión en Redes SDN/NFV

Los principios de SDN y NFV ofrecen diversas ventajas sobre arquitecturas tradicionales de red. Diversos consorcios conformados por operadores de red, universidades, centros de investigación, proveedores de servicios, entre otros, han concentrado sus esfuerzos en el desarrollo de arquitecturas de gestión innovadoras sobre entornos de red virtualizados. En la Tabla 12.1 se describen proyectos relevantes de gestión basados en SDN y NFV.

Tabla 12.1: Proyectos para la gestión de red basados en SDN/NFV

Proyecto	Dominio	Descripción	Escenario de Aplicación
CROWD [cro17]	SDN, SON	Este proyecto tiene el objetivo de aumentar la capacidad en densidad de las redes de acceso inalámbrico heterogéneas. Asimismo, se centra en garantizar la QoE de los usuarios móviles, la optimización de los mecanismos MAC y el consumo de energía. De esta forma, se mejora la gestión del tráfico en redes inalámbricas con alta densidad.	Gestión de Tráfico
5G-NORMA [nor17]	SDN, NFV	Este proyecto se centra en proporcionar la capacidad de adaptación de un recurso de manera eficiente. El framework gestiona las fluctuaciones en la demanda de tráfico por medio de un portafolio de aplicaciones de servicios. Las nuevas funciones de red ofrecen el soporte eficiente de recursos en distintos escenarios, y ayudan a incrementar la eficiencia energética.	Escenario Multi-servicio, Escenario Multi-tenant
MCN [mcn17]	SDN	El proyecto se centra en la mejora del procesamiento del tráfico mediante la separación entre hardware de radio y hardware de reenvío de paquetes.	Entorno SDN
UNIFY [uni17]	SDN, NFV	El proyecto tiene como objetivo desarrollar una plataforma de creación automatizada y dinámica de servicios a través de la creación de un modelo de servicios y un lenguaje de relación de servicios. El proyecto permitirá el despliegue dinámico y automático de los servicios en recursos de red, computación y de almacenamiento que se encuentran disponibles en la infraestructura. De manera similar, el orquestador incluirá algoritmos de optimización para asegurar la colocación óptima de componentes de servicio elementales a lo largo de la infraestructura.	Infraestructura. Virtualización Encadenamiento flexible de servicios. Invocación de cadena de servicios de red para proveedores
T-NOVA [tno17]	SDN, NFV	Este proyecto se centra en el despliegue de Funciones de Red como Servicio (NFaaS) sobre infraestructuras de red virtualizadas. Para este propósito, se diseña e implementa una plataforma de gestión y orquestación para la provisión automatizada, configuración, monitorización y optimización de recursos virtualizados. Además, SDN se utiliza para la gestión eficiente de la infraestructura de red.	Escenario de Alto Nivel, Escenario de concatenación de NFV.

12.3 Arquitectura SELFNET de Gestión Autónoma para Redes SDN/NFV

El proyecto SELFNET perteneciente al programa H2020 tiene el propósito de diseñar e implementar un framework de gestión autónomo que provea capacidades auto-organizativas SON en las nuevas infraestructuras móviles 5G. A través de la detección y mitigación automática de problemas comunes de red, que actualmente son tratados de forma manual por los administradores de red, SELFNET proveerá un framework capaz de reducir de forma significativa los costes operacionales y, en consecuencia, mejorar la experiencia de usuario [sel17], [NCC+16].

Mediante la integración de tecnologías innovadoras como SDN, NFV, SON, Cloud Computing, Inteligencia Artificial, QoS/QoE y conceptos de redes de nueva generación; SELFNET proveerá un sistema de gestión de red inteligente, escalable y extensible. Este framework asistirá a los operadores de red en el desempeño de tareas de gestión, tales como el despliegue automático de aplicaciones SDN/NFV que provean capacidades de monitorización y mantenimiento autónomo de la red. Directivas y políticas de gestión de alto nivel pondrán en marcha acciones que mitiguen problemas actuales o potenciales

amenazas futuras. SELFNET abordará tres principales escenarios de gestión de red: la provisión de capacidades de auto-protección (self-protection) contra ataques de red distribuidos, capacidades de auto-recuperación (self-healing) contra fallos en la red, y capacidades de auto-optimización (self-optimization) que mejoren dinámicamente el rendimiento de la red y la QoE. Estas funcionalidades provistas por SELFNET otorgarán los fundamentos para cumplir con algunos de los requierimientos de 5G, definidos por el consorcio 5G-PPP.

En este contexto, la Figura 12.1 ilustra la arquitectura SELFNET. Esta arquitectura está basada en seis capas diferenciadas con los siguientes alcances a nivel lógico: Capa de Infraestructura, Capa de Datos de Red, Capa de Control SON, Capa Autónoma SON, Capa de Gestión y Orquestación NFV y Capa de Acceso SON. En las siguientes secciones, cada capa es descrita.

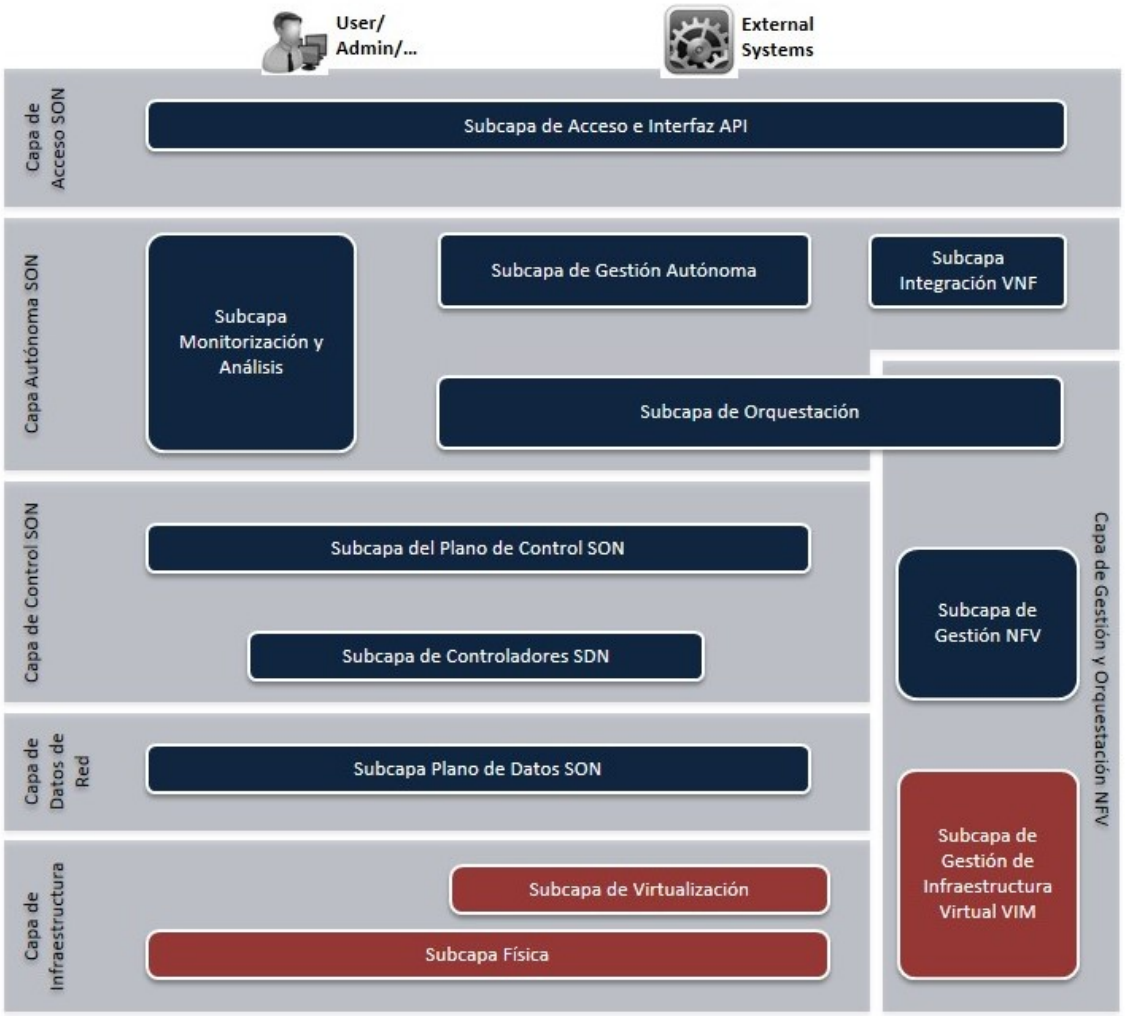


Figura 12.1: Vista general de la arquitectura SELFNET [NCC+16]

12.4 Capa de Infraestructura

Esta capa provee los recursos necesarios par la instanciación de funciones virtuales (computacionales, red, almacenamiento) y soporta los mecanismos necesarios para hacerlo. Aquello representa el componente NFVI definido en la terminología ETSI NFV [ETS13a]. Para lograr su operatividad, se definen dos subcapas: Subcapa Física y Subcapa de Virtualización.

12.4.1 Subcapa Física

La Subcapa Física incluye los recursos físicos requeridos para proveer capacidades computacionales, de red y de almacenamiento sobre hardware bare metal. Debido a que SELFNET está diseñado para operar sobre 5G, los elementos físicos siguen una arquitectura de borde (mobile edge) en la que los operadores pueden desplegar servicios operacionales y de gestión. El modelo MEC propuesto por ETSI [PNC+14] se ilustra en la Figura 12.2. Dicho modelo propone que los nodos de borde (edge) se encuentren geográficamente separados del centro de datos. De este modo, ciertos servicios pueden ser desplegados tanto cerca del usuario, así como en el centro de datos en caso de requerir alto rendimiento. Adicionalmente, la integración de despliegues de borde (como C-*Radio Access Network* (RAN)) dentro de MEC quiebra la rigidez típica y facilita la personalización de servicios. Asimismo, se considera que la conectividad entre sus elementos permite capacidades de virtualización alineadas con los avances de 5G.

12.4.2 Subcapa de Virtualización

La Subcapa de Virtualización posibilita la compartición de los recursos disponibles entre distintos usuarios o servicios. De este modo, se otorga diversas ventajas, tales como el aislamiento, confiabilidad, adaptabilidad y control de los recursos. Sin embargo, la principal desventaja incluye la penalización en el rendimiento producto de las tareas de virtualización. En este contexto, avances recientes en el ámbito de la virtualización cumplen con las expectativas de las infraestructuras 5G con respecto al rendimiento de los recursos virtualizados en entradas y salidas (I/O) [AGHE+15]. En otras palabras, la penalización en el rendimiento por el uso de la virtualización puede considerarse despreciable para los dispositivos modernos. En SELFNET, las capas de virtualización incluyen el uso de switches virtuales para la conexión de máquinas virtuales desplegadas sobre recursos físicos.

12.5 Capa de Datos de Red

En esta capa, las distintas funciones de red son situadas e interconectadas sobre una determinada topología. Las funciones de red NFs incluyen las instancias requeridas para la normal operatividad de la infraestructura virtual así como aquellas creadas por SELFNET como parte de las funcionalidades SON. Debido a que los bordes (edges) y centro de datos

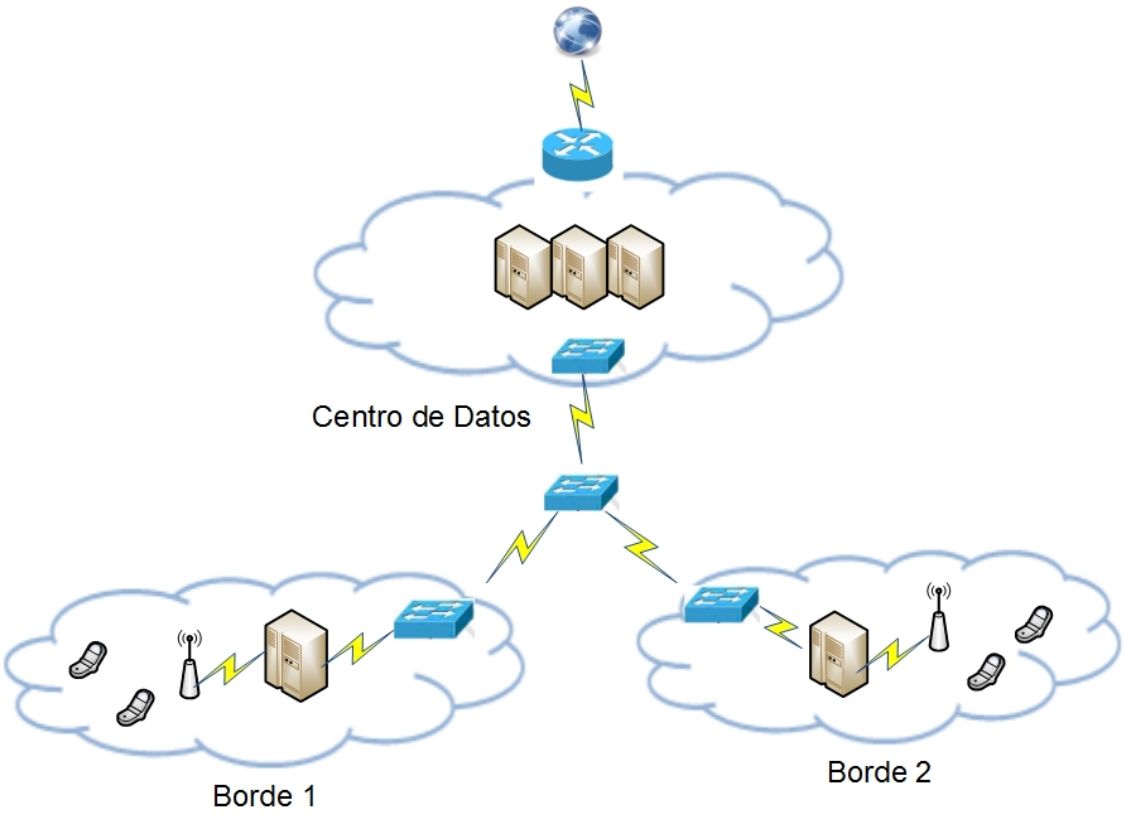


Figura 12.2: Capa física de una infraestructura MEC para arquitecturas 5G

están completamente virtualizados, las NFs pueden ser dinámicamente asignadas en ambas ubicaciones.

La Capa de Datos de Red también proporciona soporte multi-dominio que posibilita la compartición de recursos entre distintos dominios, cada cual con su propio sistema de gestión según su modelo de negocio. En las arquitecturas 5G, los recursos pueden ser adquiridos por una alianza entre operadores de telecomunicaciones, y posteriormente son compartidos de acuerdo a sus necesidades. En este contexto, un administrador particular no es capaz de gestionar los recursos de otro operador, o interceptar el tráfico provisto por otro operador.

12.6 Capa de Control SON

Esta capa incluye los elementos responsables de la recolección de datos desde distintas fuentes virtualizadas (sensores SON) y las funciones que ejecutan acciones en la red (actuadores SON). Los sensores y actuadores SON son controlados por la Capa Autónoma SON, la cuál otorga autonomía a la red. De forma similar, la Capa de Control SON interactúa con el plano de control SDN. En otras palabras, traduce políticas globales de gestión autónoma en configuraciones específicas para los elementos de la red.

12.6.1 Subcapa de Controladores SDN

La Subcapa de Controladores SDN representa un controlador lógicamente centralizado (plano de control SDN). El cual gestiona los elementos de la red y controla las funciones que se ejecutan en dichos elementos. El controlador SDN utiliza una interfaz para configurar las reglas que deben ser ejecutadas en los elementos de red. De este modo, el tráfico que circula a través de dichos dispositivos puede ser dinámicamente modificado. Adicionalmente, el controlador puede modificar funciones adicionales de los dispositivos, por ejemplo protocolos como OpenFlow, OFConfig, NETCONF . Los distintos servicios que ofrece un controlador SDN son desplegados a través de Aplicaciones o SDN-Apps. Ejemplos de SDN-Apps incluyen protocolos de encaminamiento, protocolos de etiquetado o filtrado. En la interfaz superior (northbound), el controlador SDN proporciona una API que permite la gestión, configuración y monitorización remota del comportamiento de la red.

12.6.2 Subcapa del Plano de Control SON

La Subcapa del Plano de Control SON instancia las distintas funciones de red NF que se ejecutan en la infraestructura virtualizada. En la arquitectura SELFNET existen dos tipos de NFs: Sensores SON y Actuadores SON.

- **Sensores SON**. Recopilan datos relacionados con las actividades de red. La información recolectada incluye métricas relacionadas con el tráfico global (ej. estado de los enlaces, ancho de banda) o métricas específicas (ej. DPI, QoS de una transmisión de video relacionado a un flujo de datos específico). Los operadores y proveedores de servicios pueden desarrollar distintos sensores de acuerdo a sus necesidades.

- **Actuadores SON**. Ejecutan un conjunto de acciones específicas sobre el tráfico que circula en la red. Las acciones dependen de la aplicación desarrollada por el proveedor de servicios. Por ejemplo, si el sistema detecta un ataque DDoS, un actuador SON puede automáticamente bloquear el origen de dicho ataque. A su vez, si el sistema detecta degradación de la calidad de servicio, otro actuador SON puede optimizar los flujos de tráfico incrementando la prioridad o el ancho de banda.

12.7 Capa Autónoma SON

Esta capa es responsable de proporcionar inteligencia a la red. La información recolectada por los sensores es usada para diagnosticar el estado de la red. Luego, las acciones para alcanzar los objetivos del sistema son determinadas y ejecutadas. Los principales componentes de la Capa Autónoma SON se describen a continuación.

12.7.1 Subcapa de Monitorización y Análisis

Esta subcapa recolecta los datos provistos por los sensores, los cuales son agregados y correlacionados para extraer información relevante. El analizador usa dicha información

para detectar situaciones sospechosas en la red (detección de una botnet, degradación de QoS/QoE, ataques DDoS, caída de enlaces). El proceso completo se organiza en tres fases: Monitorización y Descubrimiento, Agregación y Correlación, y Análisis.

- **Monitorización y Descubrimiento**. Se encarga de recolectar los datos enviados por los sensores SON. Para este propósito, cuando un nuevo sensor es instanciado, se recibe una notificación y los detalles de dicha instanciación para establecer una conexión que permita luego recibir las métricas correspondientes. Adicionalmente, se recibe la información provista por las subcapas física y virtual. Luego, la información es almacenada en una base de datos con el fin de facilitar su procesamiento en capas superiores.

- **Agregación y Correlación**. Lleva a cabo la correlación y agregación de la información almacenada en la base de datos de monitorización. Este proceso involucra acciones adicionales, como la normalización de datos, verificación y eliminación de información redundante. Al finalizar esta etapa, sólo la información relevante será procesada por el módulo de Análisis.

- **Análisis**. Su objetivo principal es el análisis exhaustivo de la información relevante proporcionada por la capa de Agregación y Correlación. El análisis incluye la predicción de futuros problemas de red inferidos a partir de métricas HoN. Para este propósito, se aprovechan las ventajas de diversos algoritmos de predicción, reconocimiento de patrones y técnicas de big data. Los valores inferidos permiten la aplicación de acciones preventivas y correctivas en el sistema. En esta etapa, los eventos son enviados a la Subcapa de Gestión Autónoma para establecer las correspondientes acciones en la red.

12.7.2 Integración VNF

Actúa como un repositorio de diferentes funciones de red (NFs). En esta subcapa, las funciones de red disponibles son almacenadas para que sean desplegadas como parte de una acción preventiva o correctiva. A su vez, los proveedores de servicios pueden diseñar, crear y actualizar sus propias aplicaciones. En este contexto, el encapsulamiento de NFs sigue las recomendaciones del framework ETSI MANO [ETS14]. Consecuentemente, el gestor de NFV (VNFM) es un componente clave para todo el ciclo de vida de los sensores y actuadores SON. El ciclo de vida VNFM expone un conjunto común de primitivas para la instanciación, configuración, reconfiguración y eliminación automática de las distintas VNFs. Una API común facilita a los proveedores el fácil diseño y desarrollo de sus soluciones. Una vez que la solución NF es publicada en el repositorio (onboarding), el Gestor Autónomo utiliza dichas funcionalidades para proveer un nuevo servicio (sensor/actuador).

12.7.3 Subcapa de Gestión Autónoma

Este componente usa distintos algoritmos para diagnosticar la causa de un problema de red sobre la base de las métricas HoN provistas por el Analizador. Una vez que la causa es

detectada, el Gestor Autónomo usa las NFs disponibles que ofrece el repositorio de VNFs para decidir la mejor estrategia de reacción, o una contramedida (por ej. El despliegue de un nuevo balanceador de carga, cortafuegos o DPI). Luego, las acciones decididas son comunicadas a la Capa de Gestión y Orquestación NFV. Las tareas relacionadas con la Gestión Autónoma están agrupadas en tres módulos.

- **Diagnosticador**. Este elemento diagnostica la causa de los problemas de red a partir de la información proporcionada por la Subcapa de Monitorización y Análisis (topología, datos de sensores, métricas HoN). Asimismo, aprovecha las ventajas de algoritmos estocásticos, inteligencia artificial y minería de datos para determinar el origen del problema. Finalmente, la causa es notificada al submódulo de Toma de Decisiones.
- **Toma de Decisiones**. Recoge la información procedente del Diagnosticador y decide el conjunto de acciones preventivas o correctivas a ser desplegadas con el fin de mitigar los problemas de red detectados. Del mismo modo, este componente aprovecha también la integración de algoritmos de inteligencia artificial para determinar las respuestas o tácticas a ser desplegadas. Las acciones tomadas son notificadas al Ejecutor de Acciones.
- **Ejecutor de Acciones**. Proporciona un conjunto de acciones consistentes a ser ejecutadas en la infraestructura. En otras palabras, valida, organiza y refina las tácticas para evitar conflictos, duplicidad y orden incoherente de las acciones. Al finalizar esta etapa, una descripción de alto nivel de la localización, tipo de actuador SON y parámetros de configuración asociados son transferidos al Orquestador.

12.8 Capa de Gestión y Orquestación NFV

Esta capa es responsable del control y concatenación de las distintas NFs en la infraestructura virtualizada. La arquitectura sigue las recomendaciones [ETS14] y, consecuentemente, está compuesta de: Orquestación, Gestión de VNFs y Gestión de Infraestructura Virtualizada VIM. Tal como se describe en la sección 12.7.2, las operaciones de Gestión de VNFs son parcialmente desarrolladas en la capa de integración VNF. El resto de operaciones se describe a continuación:

- **Subcapa de Orquestación y Gestión NFV**. Es responsable de recibir el conjunto de acciones del Gestor Autónomo y orquestar las correspondientes funciones de red sobre los recursos virtuales disponibles. La coordinación y programación de la ejecución de diferentes acciones se realiza mediante la interacción con el Gestor de Infraestructura Virtual.
- **Subcapa de Gestión de Infraestructura Virtual (VIM)**. Se encarga de organizar y proporcionar los recursos virtuales para la instanciación de las diferentes funciones de red. El VIM interactúa con la infraestructura física y virtual para asegurar la disponibilidad de recursos, y realizar el despliegue automático de servicios.

12.9 Capa de Acceso SON

Esta capa proporciona una interfaz atractiva e intuitiva que proporciona diferentes capacidades de monitorización y operación dependiendo de los niveles de privilegios de los usuarios. De esta forma, los usuarios pueden comprobar el estado actual de las operaciones en SELFNET. Asimismo, la API de acceso registra los sensores y actuadores SON actualmente desplegados en SELFNET, así como las sesiones iniciadas y mensajes, que permiten una visión más amplia del estado de la red. Esta interfaz es usada por actores externos como Sistemas de Soporte Empresarial (BSS) o Sistemas de Soporte Operacional (OSS).

Como se describió en las secciones anteriores, SELFNET pretende ser una solución independiente y autónoma que actúe en la mitigación o resolución de problemas de red sin intervención de los administradores de red. De este modo, la Capa de Acceso SON proporciona también a los usuarios el estudio de las acciones ejecutadas por SELFNET, posibilitando la validación de acciones correctivas de las aplicaciones.

12.10 Resumen

Este capítulo hace un resumen de los avances en auto-gestión de redes basadas en los principios SDN/NFV. Luego, el proyecto SELFNET es presentado. A continuación, las distintas capas y subcapas del framework SELFNET son descritas. Las capas de Infraestructura, Datos de Red, Control SON, Capa Autónoma SON, Gestión y Orquestación NFV y Capa de Acceso SON son presentadas.

Capítulo 13

SELFNET Monitorización y Descubrimiento en Redes SDN/NFV

El capítulo presenta la arquitectura de monitorización y descubrimiento para redes SDN/NFV basadas en la propuesta SELFNET. El diseño toma en cuenta los requisitos de escalabilidad y flexibilidad visionados en la futura infraestructura de redes móviles 5G. En este sentido, la presente arquitectura se centra en la recolección y almacenamiento de información perteneciente a elementos físicos, virtuales, entornos en la nube, métricas a nivel de flujo, tráfico SDN y sensores. De igual manera, la arquitectura permite obtener información de métricas pertenecientes a diferentes entornos y casos de uso, ya que cada fuente de información es tomada como una fuente genérica, habilitando la ejecución de operaciones de correlación y agregación. El presente diseño permite la obtención y almacenamiento de información de subcapas de la arquitectura SELFNET, el cual incluye funciones virtuales desplegadas sobre la infraestructura virtual, también conocidos como sensores SELFNET.

El presente capitulo se encuentra organizado de la siguiente manera: la Sección 13.1 presenta el capítulo. La Sección 13.2 describe la arquitectura de monitorización y descubrimiento SELFNET. La Sección 13.3 define los diagramas de flujo de la arquitectura propuesta. Del mismo modo, la Sección 13.4 presenta detalles de la implementación de la arquitectura. Finalmente, la Sección 13.5 resume el presente capítulo.

13.1 Introducción

El proyecto SELFNET tiene entre sus objetivos el monitorizar el estado de la red y los servicios disponibles con el fin de brindar capacidades auto-organizativas de alto nivel. En este sentido, la presente arquitectura de monitorización y descubrimiento permite la recopilación de datos de cinco fuentes diferentes. Las fuentes de información incluyen dispositivos físicos y virtuales, entornos en la nube, métricas a nivel de flujo de datos, tráfico SDN y métricas de sensores.

La arquitectura de monitorización y descubrimiento SELFNET presenta ventajas adicionales con respecto a las actuales soluciones de monitorización basados en SDN/NFV. La propuesta presentada en [YCY+15] usa los principios de SDN para virtualizar un dispositivo de red y monitorizar el tráfico sin la necesidad de replicación de puertos. Sin embargo, el sistema no es capaz de recoger métricas a otros niveles (elementos virtuales, sensores, etc.). De igual manera, NetAnlytics [LTRW16] puede monitorizar eficientemente el plano de datos a nivel de flujo y usa NFV para instanciar las funciones de red. Sin embargo, las métricas recogidas se enfocan únicamente a nivel de flujo y no permiten personalización de múltiples fuentes. La arquitectura presentada en [GKM+16] permite la monitorización del gestor de infraestructura virtual VIM. El mismo incluye varios agentes para recolectar métricas de funciones virtuales (VNFs). Sin embargo, la propuesta se presenta como un componente interno del VIM y no incluye el acceso a fuentes adicionales (ej. métricas de entorno físico), así como tareas de agregación y correlación. En [SWL+16] se propone una arquitectura de referencia para facilitar las tareas de ingeniería de tráfico en redes SDN. La propuesta está compuesta por un nivel de monitorización de tráfico (a nivel de red) y otro nivel de gestión de tráfico (QoS, balanceo de carga). Sin embargo, la propuesta no incluye la flexibilidad brindada por las bases de NFV.

La arquitectura de monitorización y descubrimiento SELFNET senta las bases de un modelo abierto a múltiples métricas de bajo nivel. De este modo, el usuario será capaz de reorganizar la información recibida en función a diferentes criterios, tales como instancias virtuales por dominio, métricas de instancias virtuales, métricas de parámetros físicos, estadísticas a nivel de flujo de red y métricas de agentes o sensores. Como resultado, la presente arquitectura facilita no sólo el proceso de consulta de la información, sino también la gestión de grandes cantidades de información de fuentes heterogéneas de datos. En un siguiente nivel, las métricas disponibles en cada capa podrán ser agregadas y correlacionadas para brindar información relevante del estado de la red, tales como la relación entre estancias virtuales y su respectivo equipo físico, instancias virtuales y físicas en el cual se encuentra ejecutándose un sensor, información relacionada a dispositivos LTE, entre otros.

13.2 SELFNET Monitorización y Descubrimiento

Uno de los principales retos de la solución propuesta por SELNET es la monitorización y descubrimiento de las diferentes métricas generadas por las capas inferiores de la infraestructura. Las métricas de interés pueden incluir información de bajo nivel, indicadores clave de rendimiento KPI y métricas del estado de la red de alto nivel HoN, por medio de los cuales los operadores pueden tener un conocimiento y visión clara del estado de la red. Esta propuesta difiere considerablemente de los esquemas tradicionales, en donde la localización de los agentes de monitorización y la información enviada es siempre estática. Por su parte, la presente arquitectura recibe información de funciones virtuales de redes que pueden ser ubicados de forma dinámica en diferentes zonas de la red. Del mismo modo, cada agente puede enviar información de diferente naturaleza en función del objetivo particular del sensor. Asimismo, la arquitectura recibe información

de fuentes correspondientes a métricas de otras subcapas de la infraestructura. La Figura 13.1 describe las diferentes interfaces usadas, tanto para recibir información como para enviar los resultados del analizador.

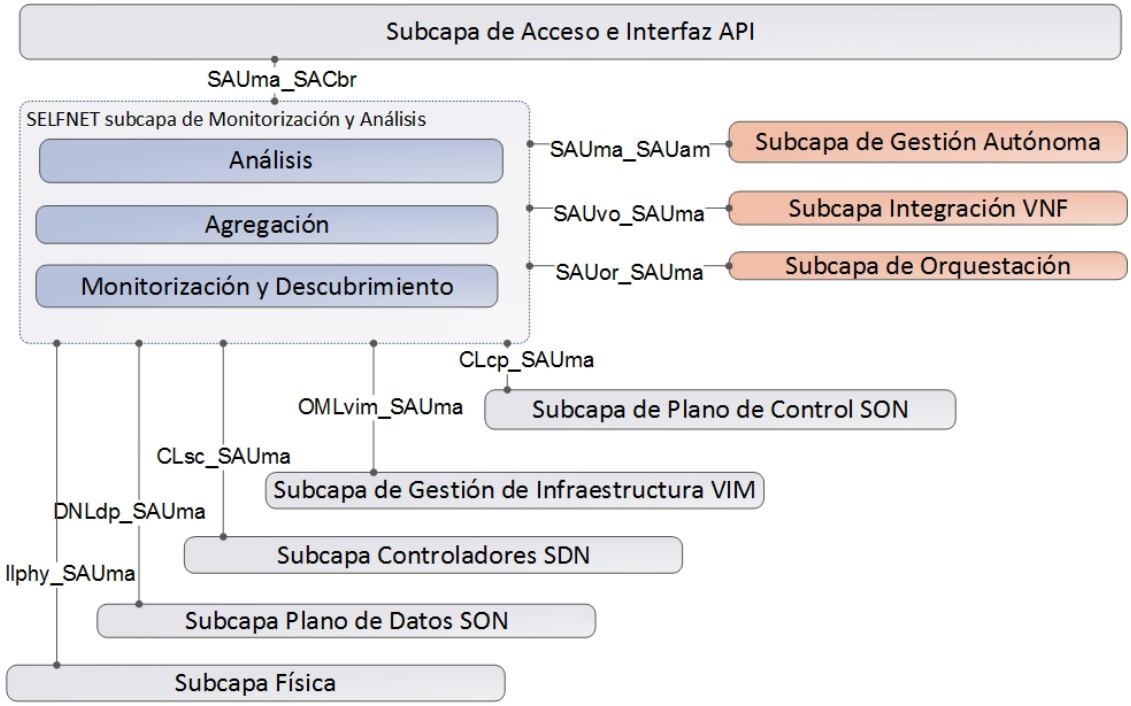


Figura 13.1: Interfaces conectadas al módulo de monitorización y análisis

El módulo de monitorización y análisis interactúa con otras subcapas a travez de diferentes interfaces. La Tabla 13.1 describe el nombre, fuente, destino y la información enviada. Por un lado, diferentes fuentes envían su información para ser procesada y analizada. Por otro lado, el resultado de las tareas de agregación y análisis es enviado a la subcapa autónoma. Finalmente, el estado de la red es transmitido a la subcapa de acceso e interfaz API.

13.2.1 Arquitectura de Alto Nivel

El principal objetivo de la subcapa de Monitorización y Análisis es proveer los componentes necesarios para analizar el estado de la red mediante la eficiente recolección de información, el cual proviene de diferentes fuentes ubicadas en la infraestructura de la red. Con este propósito, el módulo de monitorización y análisis ha sido dividido en tres subcapas: monitorización y descubrimiento, agregación y análisis. El módulo de análisis se encarga de encontrar problemas en la red, tanto los problemas actuales, así como los posibles futuros eventos. Dichos resultados de alto nivel son conocidos también como estado de salud de la red HoN y son enviados a la subcapa de gestión autonóma. Las métricas HoN son derivadas de métricas de bajo nivel que fueron agregadas y correlacionadas por el módulo de agregación. Por su parte, las métricas de bajo nivel son provistas por el módulo de monitorización y descubrimiento.

Tabla 13.1: Interfaces de monitorización y análisis

Nombre	Fuente	Destino	Información transmitida
ILphy_SAUma	Física	Monitorización y Análisis	Métricas de variables físicas
OMLvim_SAUma	VIM	Monitorización y Análisis	Recursos virtuales
DNLdp_SAUma	Plano de Datos SON	Monitorización y Análisis	Métricas del plano de datos
CLsc_SAUma	Controladores SDN	Monitorización y Análisis	Métricas del plano de control SDN
SAUvo_SAUma	Integración VNF	Monitorización y Análisis	Información descriptiva de los sensors NFV
SAUor_SAUma	Orquestación	Monitorización y Análisis	Instanciación de sensors NFV
SAUma_SAUam	Monitorización y Análisis	Gestión Autónoma	Resumen del estado actual de la red a alto nivel, así como valores de predicción de las posibles amenazas de la red
SAUma_SACbr	Monitorización y Análisis	Acceso e Interfaz API	Estado actual de las tareas realizadas en la capa de monitorización y análisis

La arquitectura se encuentra dividida en bloques funcionales, tal como se describe en la Figura 13.2. En la zona inferior se localizan las diferentes fuentes de datos. La arquitectura introduce el concepto de fuente de datos o "data source" como componente funcional que permite la trasferencia de datos desde el correspondiente elemento de monitorización. La fuente de datos es capaz de escoger el método de recolección de información desde los puntos de monitoreo, ya sea "polling" o "pushing". De este modo, la fuente de datos implementará la interfaz particular para comunicarse con el punto de monitoreo. Por ejemplo, la fuente de datos puede ser configurado para via REST acceder al sensor, que puede ser una función virtual instanciada en la infraestructura. De este modo, la información del sensor puede ser enviado al módulo de monitorización. La función de descubrimiento permite a la arquitectura detectar las nuevas instancias de monitorización, su configuración y parámetros de comunicación para el envío de datos.

13.2.2 Descriptores de Sensores Virtuales

Los descriptores de sensores virtuales tienen la tarea de recibir, desglosar y guardar la información relacionada con el tipo de sensores disponibles en el catálogo SELFNET. Este elemento interactúa con otros 2 componentes de SELFNET: Subcapa Integración VNF y Gestor de Fuentes de Datos.

Los descriptores usan la interfaz SAUvo_SAUma para conectarse a la subcapa integración VNF. De este modo, los descriptores son notificados cuando un nuevo sensor VNF es incorporado en el catálogo del sistema SELFNET, así como en el caso de su actualización o eliminación. En otras palabras, cada vez que existe algún cambio en los sensores VNF, el módulo de monitorización es notificado y se registran dichos cambios internamente. Los descriptores registran dichos cambios en una base de datos interna, también conocido como catálogo local de sensores. Una interfaz interna permite al gestor de fuentes de datos tener disponible la información relevante de los potenciales sensores que pueden ser instanciados en la infraestructura. Dicha información incluye el tipo de sensor,

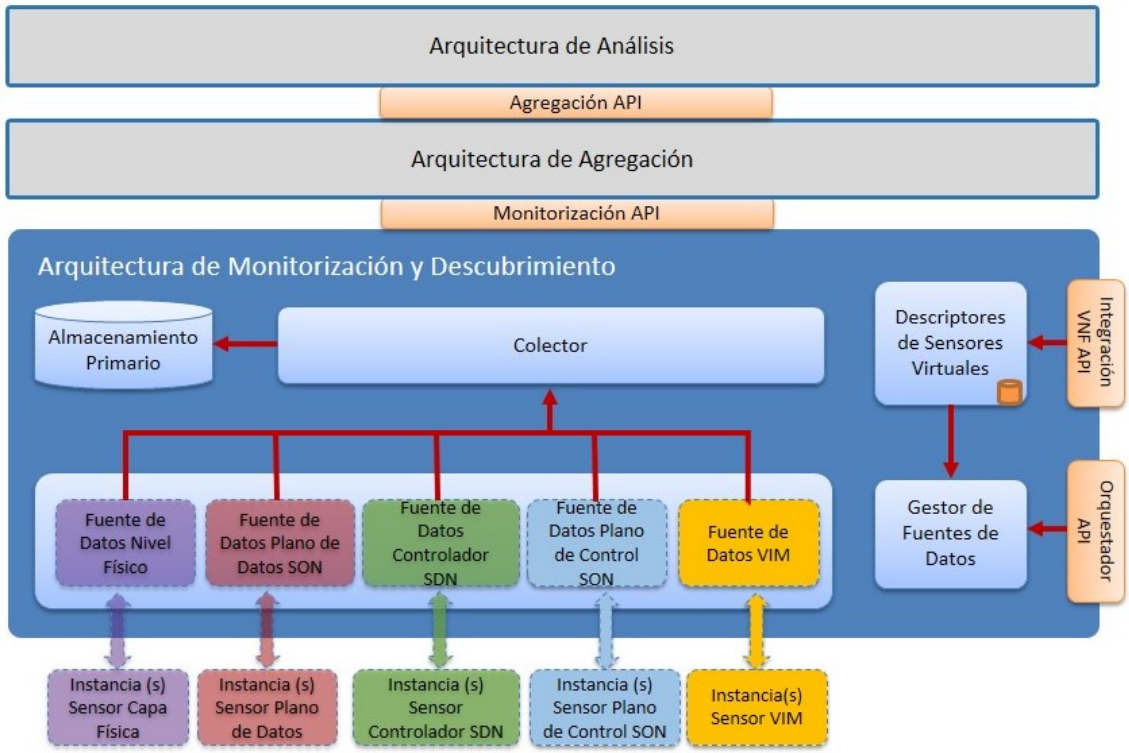


Figura 13.2: Arquitectura de monitorización y descubrimiento

el método de comunicación y las métricas que pueden ser recolectadas. Sin embargo, no contiene la información relacionada con una instancia específica.

13.2.3 Gestor de Fuentes de Datos

Dentro del diseño de la arquitectura de monitorización y descubrimiento, una fuente de datos es un componente funcional encargado de establecer el camino entre la fuente de información, recopilar la información enviada por los sensores y enviar dicha información al colector de datos. Con este objetivo, la fuente de datos requiere la información relacionada con una instancia particular del sensor, como por ejemplo la dirección IP, el protocolo de comunicación, la frecuencia de muestreo, entre otros. Esta información es asignada en el momento que el orquestador SELFNET instancia una función virtual NFV en la infraestructura. En este sentido, el orquestador notifica al gestor de fuente de datos la información relevante para configurar adecuadamente una fuente de datos particular. En esta etapa, la fuente de datos se encuentra lista para recoger información de una instancia determinada.

13.2.4 Instancias de Fuentes de Datos

Este apartado resume las diferentes fuentes de datos y las métricas que dichas fuentes envían. La Figura 13.3 muestra la estructura común compartida por las instancias, la fuente de datos y el respectivo flujo de información. Una vez que el gestor de instancias inicia una fuente de datos específica, dicha fuente recibe la configuración e inicia la

recolección de métricas. La información circula hacia el colector y se guarda en la base de datos de almacenamiento primario.

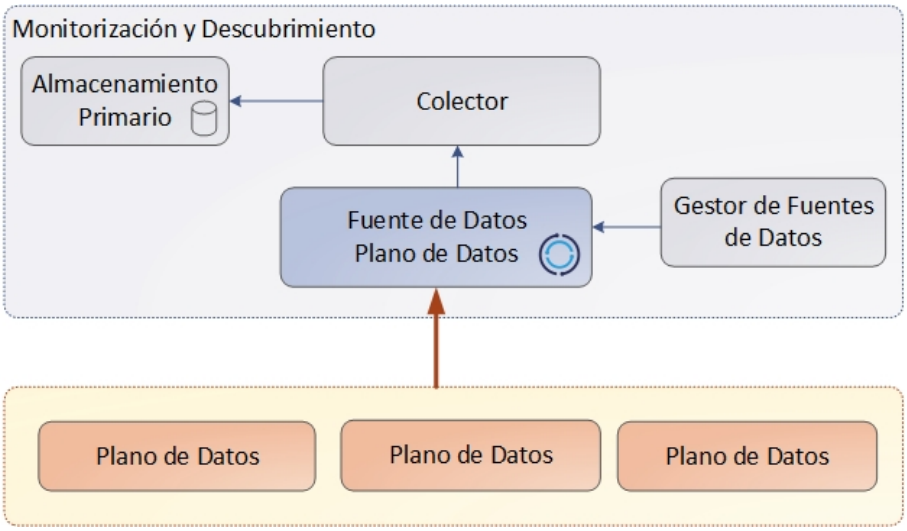


Figura 13.3: Fuentes de datos

La Tabla 13.2 describe las instancias y métricas enviadas por las diferentes fuentes de datos.

Tabla 13.2: Diferentes instancias de fuente de datos

Instancia	Descripción
Capa Fisica	Métricas relacionadas con el equipamiento físico en el cual se instanciarán los elementos virtuales. También se incluyen las métricas de los elementos de red que habilitan la comunicación de los diferentes componentes de la infraestructura.
Gestión de Infraestructura Virtual VIM	Métricas relacionadas con los recursos virtuales y entorno en la nube que se encuentran ejecutándose en la capa física. El mismo incluye métricas de dominios, entornos virtuales de red, ubicación de los recursos virtuales.
Controlador SDN	La fuente brinda información de las métricas obtenidas desde la perspectiva de un controlador SDN. En otras palabras, el controlador puede inferir las estadísticas de la red basados en la información transmitida de la interfaz entre el plano de datos y control (e.j. OpenFlow).
Plano de Datos	La fuente recolecta información relacionada con la información de red de bajo nivel. Se ha determinado que la información a nivel de flujo de datos es la solución más apropiada para determinar el estado de la red. Métricas a nivel de flujo de información resaltan las características más relevantes y reducen la cantidad de información transmitida.
Plano de Control SON	Métricas particulares relacionadas a un sensor NFV específico e instanciado sobre la infraestructura virtual.

13.2.5 Colector

El colector se define como el componente funcional encargado de escuchar las métricas y eventos enviados por las diferentes fuentes de información. De igual manera, transforma y guarda la información en la base de datos de almacenamiento primario. El colector usa un sistema autoconfigurable, el cual puede contener plugins que permitan el almacenamiento en diferentes tecnologías de base de datos.

13.2.6 Almacenamiento Primario

El almacenamiento de métricas y eventos presenta múltiples retos. La información recibida por medio del método "polling" tiene un comportamiento regular y predecible. Por su parte, el método "pushing" tiene un comportamiento impredecible. Por este motivo, la arquitectura propone el uso de bases de datos diferentes en función el tipo de información. Las métricas comunes pueden ser almacenados usando series temporales en la base de datos TSDB, debido a que las mismas son optimizadas para manejar datos temporales y los arreglos pueden ser indexados por tiempo (marca o rango de tiempo). Por su parte, los eventos podrían ser almacenados en una base de datos NoSQL cuyo objetivo es el almacenamiento de información no estructurada en un periodo corto de tiempo.

13.2.7 Interfaz Superior

La presente arquitectura de monitorización y descubrimiento expone múltiples interfaces para interconectarse con otros módulos de SELFNET. Sin embargo, el principal receptor de la información recolectada por monitorización es el módulo de agregación. Con este motivo, la interfaz proporcionada es fácilmente accesible y permite acceder al almacenamiento primario. Por ejemplo, la interfaz superior podría usar una interfaz tipo REST-API el cual permita autenticación y soporte múltiples operaciones: listar las métricas disponibles, seleccionar una métrica específica, filtrar una métrica específica en un rango determinado, entre otros. Una opción deseable también incluye especificar una granularidad determinada. Por otra parte, el uso de buses de información pueden ser la opción preferida para la notificación de eventos, debido a que generalmente se requiere una respuesta rápida. Luego de la notificación, se puede almacenar los eventos para un análisis posterior.

13.3 Procesos de Monitorización y Descubrimiento

La presente sección describe los diferentes procesos realizados por la arquitectura, así como la interacción con los diferentes módulos de SELFNET.

13.3.1 Integración de Sensores

Cuando un nuevo sensor es agregado en el sistema, algunos pasos son llevados a cabo para comunicar las nuevas funcionalidades y sus características. Este proceso es conocido como integración de sensores y se describe en la Figura 13.4.

Los dos principales componentes que participan en el presente proyecto son la Subcapa Integración VNF y los descriptores, tal como se explica en la Tabla 13.3.

13.3.2 Instanciación de Sensores

La instanciación de sensores tiene un impacto directo en la función de monitorización y descubrimiento. Una vez que un sensor se encuentra instanciado, dicho elemento inicia la recolección de información y dicha información debe ser enviado de manera eficiente al

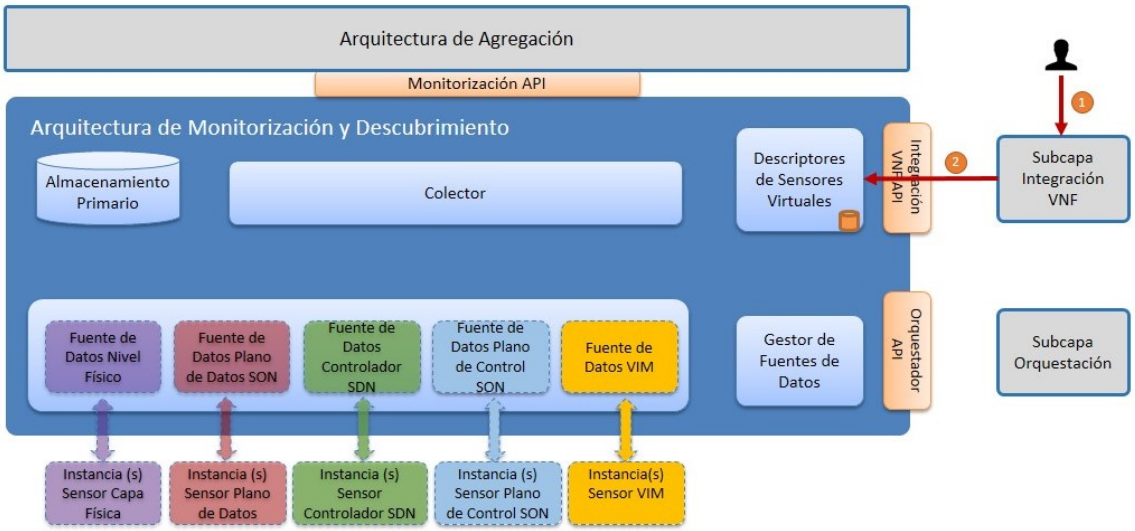


Figura 13.4: Proceso de integración de sensores

Tabla 13.3: Resumen del proceso de integración de sensores

Secuencia	Componente Fuente	Componente Destino	Descripción
1	Subcapa de Acceso e Interfaz API	Subcapa Integración VNF	Un Nuevo sensor es registrado en el sistema. El proceso es iniciado a través de la interfaz gráfica o por medio de una interfaz con un agente externo a SELFNET.
2	Subcapa Integración VNF	Virtual Sensors Descriptors	Una vez que el sensor es exitosamente registrado en el sistema por la capa de registro (onboarding), dicho evento es publicado en un mensaje de datos y todos los subscriptores recibirán dicha notificación. A dicha notificación se adjunta un archivo descriptor que contiene información y atributos relevantes del sensor.

módulo de monitorización. Por esta razón, una fuente de datos es agregado y configurado en la arquitectura de monitorización. El proceso de instanciación de sensores es descrito en la Figura 13.6.

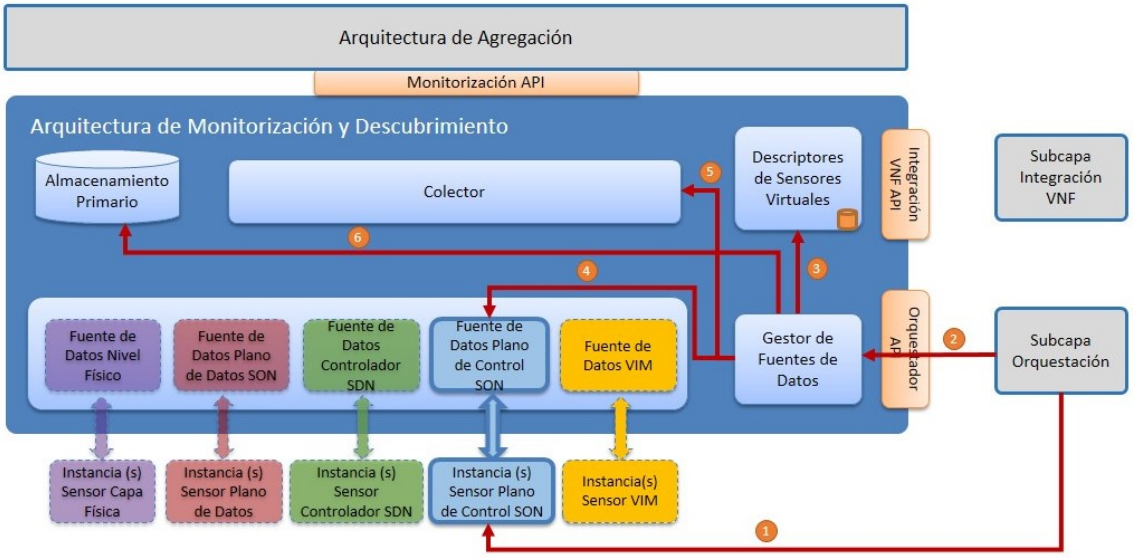


Figura 13.5: Proceso de instanciación de sensores

El presente proceso es explicado en la Tabla 13.4 y describe el proceso de una instancia localizada en el plano de control. Sin embargo, todas las otras fuentes de datos siguen un procedimiento similar.

Tabla 13.4: Proceso de instanciación de sensores

Secuencia	Componente Fuente	Componente Destino	Descripción
1	Orchestration Sublayer	Instancia Sensor	Un nuevo sensor localizado por ejemplo en el plano de control es instanciado por el orquestador usando el gestor de infraestructura virtual VIM.
2	Orchestration Sublayer	Gestor de Fuentes de Datos	Cuando el sensor es desplegado en la infraestructura virtual, el orquestador envía una notificación informando el éxito de la instancia. La notificación es recibido por el gestor de fuentes de datos. El mensaje de notificación contiene la información relevante de la instancia (tipo de sensor, dirección IP, puerto, dominido ID, etc).
3	Gestor de Fuentes de Datos	Descriptores de Sensores Virtuales	Una vez recibida la notificación de instanciación, el gestor interno de fuente de datos solicita la información de las métricas a recolectar. Esta información es solicitada a los descriptores de sensores. La información incluye las métricas recibidas, tipo de sensor, método de comunicación, entre otros.
4	Gestor de Fuentes de Datos	Instancias de Sensores de Datos	Una vez que toda la información relacionada con la instancia y tipo de sensor es obtenida, el gestor de fuente de datos crea una nueva instancia de fuente y configura dicha instancia (ej. una fuente en el plano de control). Dicho elemento, una vez configurado, inicia el proceso de recolección de datos.
5	Gestor de Fuentes de Datos	Colector	De igual manera, el gestor informa al colector de la nueva fuente de información y las métricas que serán enviadas.
6	Gestor de Fuentes de Datos	Almacenamiento Primario	Finalmente, el gestor de fuentes de datos anuncia las nuevas métricas al componente de almacenamiento de las nuevas métricas que serán enviadas.

13.3.3 Proceso de Monitorización de Sensores Instanciados

Una vez que los sensores se encuentran instanciados en la infrestructura virtualizada, las métricas obtenidas son enviadas al módulo de monitorización, ya sea por el método "polling" o "pushing". La Figura 13.6 describe el flujo de información de los sensores (métricas o eventos) hacia el módulo de monitorización.

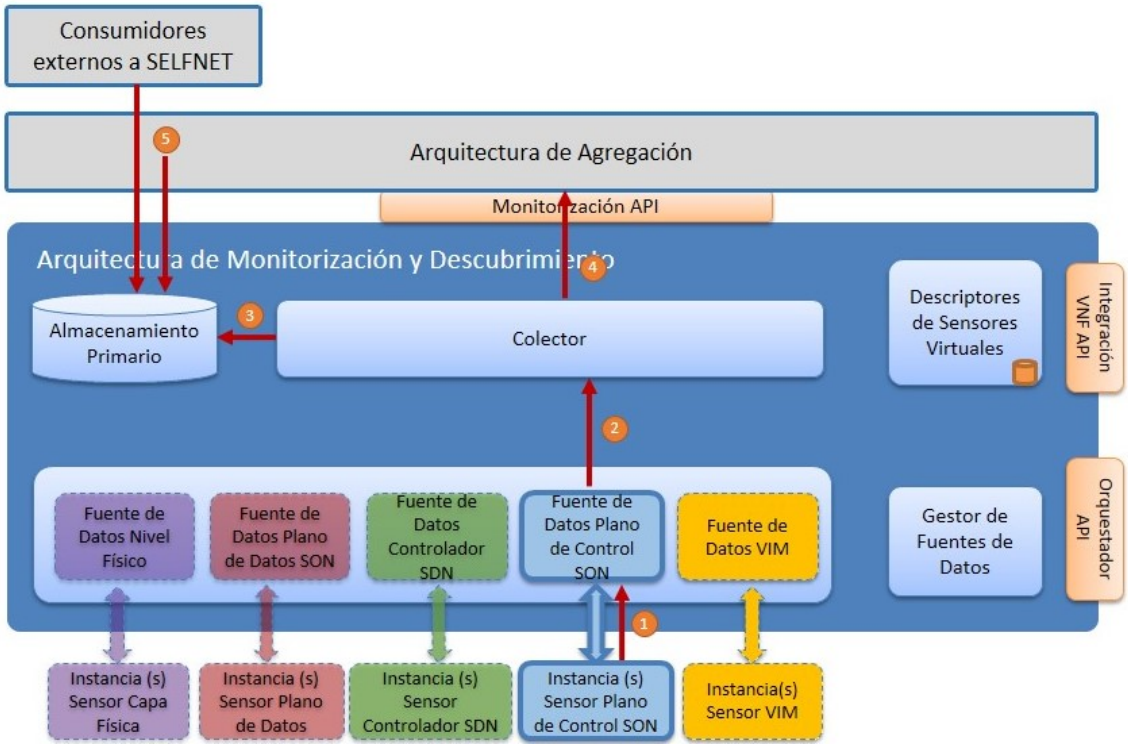


Figura 13.6: Proceso de envío de información (métricas o eventos)

El proceso se encuentra descrito en la Tabla 13.5.

13.3.4 Reconfiguración de las Características del Sensor

Mientras el sensor se encuentra funcionando, algunos parámetros pueden ser modificados sin necesidad de cambiar la instancia. Un claro ejemplo es el cambio de la frecuencia de muestreo. Por defecto, se establece un periodo de muestreo al inicio de la instancia, sin embargo dicho periodo puede ser modificado en caso de ser necesario. Esta modificación no afecta el normal funcionamiento del sensor. El procedimiento a ejecutar se encuentra reflejado en la Figura 13.7.

El proceso se encuentra detallado en la Tabla 13.6.

13.3.5 Proceso de Eliminación de Sensores

Finalmente, cuando una instancia del sensor es eliminada de la infraestructura virtual, se requiere la eliminación de la respectiva fuente de datos en el módulo de monitorización. Dicho procedimiento se encuentra reflejado en la Figura 13.8.

Tabla 13.5: Proceso de envío de métricas al modulo de monitorización

Secuencia	Componente Fuente	Componente Destino	Descripción
1	Instancia Sensor	Instancia de Sensor de Datos	La información monitorizada es recolectada por el sensor en función de su aplicación particular (DPI, sensor de video, tráfico). Dicha información es enviada a la fuente de datos del módulo de monitorización utilizando uno de los 2 métodos disponibles ('polling" o "pushing").
2	Instancia de Sensor de Datos	Colector	La fuente de recolección de datos recibe las métrics y/o eventos y los reenvía hacia el colector (bus de mensajes del colector) utilizando la interfaz apropiada.
3	Colector	Almacenamiento Primario	El Colector transforma y normaliza la información recibida y lo reenvía hacia el sistema de almacenamiento primario.
4	Colector	Aggregation Framework	Tal como se describe en el paso 3, el sistema de monitorización provee la información recolectada al módulo superior de agregación para el procesamiento de datos (agregación y correlación).
5	Consumidores externos a SELFNET	Almacenamiento Primario	Las métricas y/o eventos almacenados pueden ser accesibles por componentes externos en el caso de ser necesario. El gestor de almacenamiento prove una interfaz para el acceso externo (Almacenamiento Primario API).

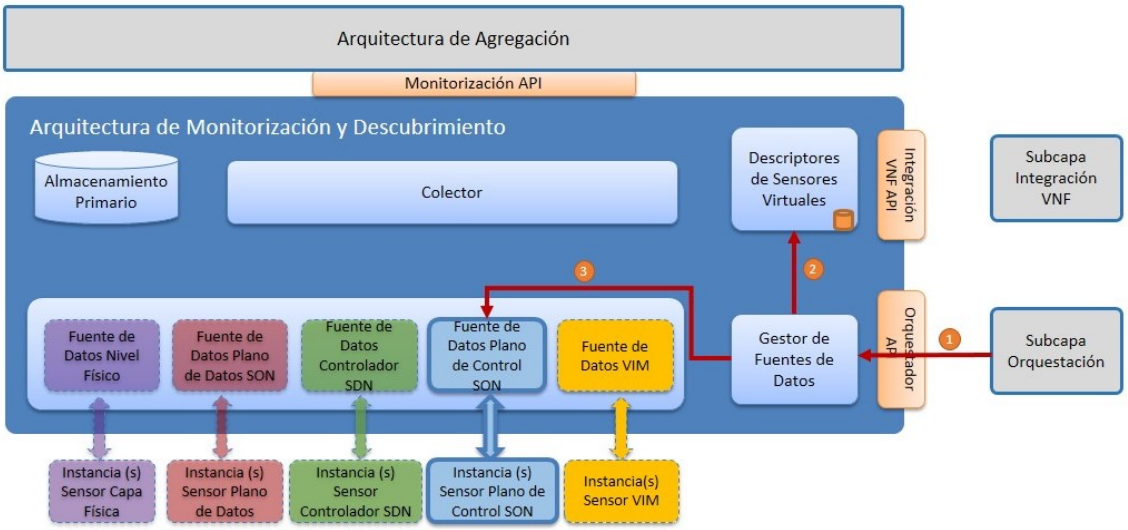


Figura 13.7: Procedimiento para la reconfiguración de la fuente de datos

El procedimiento especificado para la eliminación de un sensor se describe en la Tabla 13.7.

13.4 Implementación

La implementación de la arquitectura de monitorización presenta características particulares que permiten ofrecer las funciones descritas anteriormente. La arquitectura permite la recolección de información de múltiples fuentes a diferentes niveles (física, virtual, SDN controller, etc). Con este objetivo, multiples proyectos de código libre han sido

Tabla 13.6: Procedimiento para la reconfiguración de la fuente de datos

Secuencia	Componente Fuente	Componente Destino	Descripción
1	Subcapa Orquestación	Gestor de Fuentes de Datos	Un agente externo (que podría ser el orquestador) solicita a monitorización la modificación de las características de la fuente de datos. Por ejemplo el cambio en la frecuencia de muestreo.
2	Gestor de Fuentes de Datos	Descriptores de Sensores Virtualess	El gestor de fuentes de información solicita a los descriptores información actualizada de los sensores (en caso de ser necesario).
3	Gestor de Fuentes de Datos	Instancia de Sensor de Datos	El gestor de fuente de información solicitada al fuente de datos un cambio en la configuración, por ejemplo el cambio de la frecuencia de muestreo.

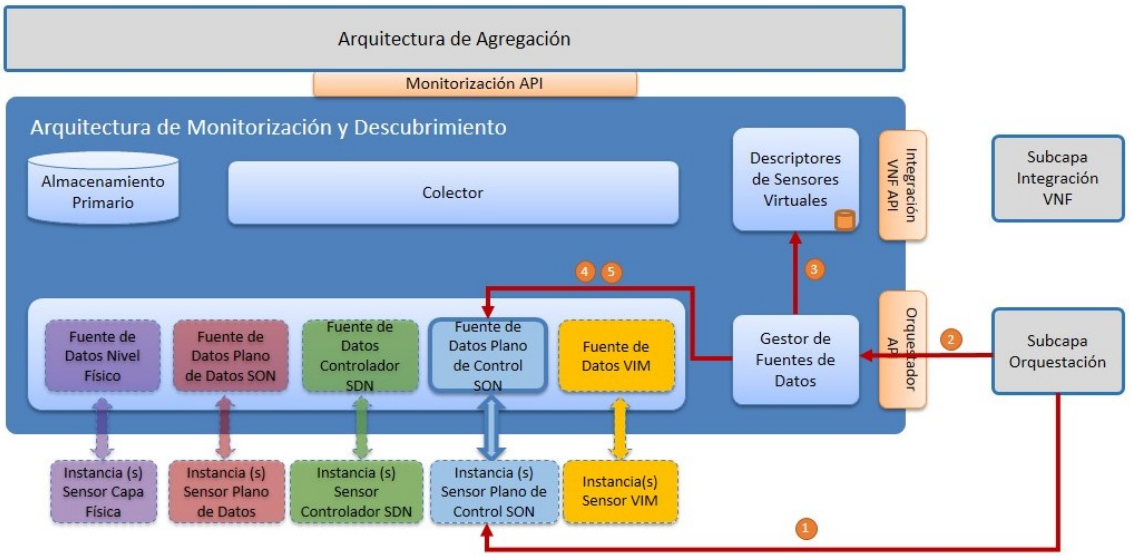


Figura 13.8: Eliminación de un sensor

Tabla 13.7: Procedimiento de eliminación de sensores

Secuencia	Componente Fuente	Componente Destino	Descripción
1	Subcapa Orquestación	Instancia Sensor	El orquestador elimina un sensor determinado, por ejemplo por una decisión del sistema de gestión autónomo.
2	Subcapa Orquestación	Gestor de Fuentes de Datos	El orquestador notifica al gestor de fuentes de información la eliminación del sensor.
3	Gestor de Fuentes de Datos	Descriptores de Sensores Virtualess	El gestor de fuentes solicita información adicional al descriptor (en caso de ser necesario).
4	Gestor de Fuentes de Datos	Instancia de Sensor de Datos	El gestor de fuentes envía una notificación a la respectiva fuente de datos que deje de recibir datos y remueva la configuración.
5	Gestor de Fuentes de Datos	Instancia de Sensor de Datos	El gestor de datos también solicita eliminar la instancia de la fuente de datos.

probados para ser incorporados en la implementación. En la presente sección se presenta la implementación del marco general y la interacción entre los componentes. De igual manera, se describen los módulos de los descriptores de sensores, el gestor de fuentes, colector y el almacenamiento.

13.4.1 Arquitectura General

La presente proyecto utiliza el projecto OpenStack [SAE12] [ope17b] y su servicio Telemetry como línea base para la implementación de la arquitectura. OpenStack es considerado como el proyecto más completo en gestión de plataformas para la nube, ya que ofrece múltiples servicios (nova, neutron, keystone, telemetry, etc.) que pueden ser integrados no sólo para plataformas tradicionales, sino en entornos SDN, NFV. De igual manera, el proyecto Telemetry [tel17a] ha sido desarrollado para facilitar la medición y monitoreo de los recursos virtuales desplegados sobre OpenStack. Telemetry gestiona diversas áreas y brinda un servicio de alarmas (Aodh), recolección de datos (ceilometer/monasca) [cei17a], base de datos de series temporales y servicio de indexación de recursos (Gnocchi) [gno17].

Ceilometer propone una arquitectura basado en plugins, el cual facilita escalabilidad, extensibilidad, personalización de métricas y seguimiento de los recursos disponibles mediante la creación de nuevos agentes. Un agente ceilometer podría recolectar información de recursos de servicios internos de OpenStack, tales como Nova o Neutron [cei17a], así como de fuentes externas como LibreNMS [lib17], ODL o Time Series Data Repository (TSDR) [tsd17]. La arquitectura Ceilometer se encuentra descrito en la Figura 13.9.

Desde la perspectiva del colector de métricas, Ceilometer provee las funcionalidades de “polling” y “pushing” para la obtención de datos. Las fuentes de información pueden ser extendidas para integrar nuevas métricas. Sin embargo, el incremento en el número de fuentes de datos y los recursos que necesitan puede ocasionar degradación exponencial del rendimiento del sistema y ocasionar problemas de escalabilidad. Con el fin de sobrellevar este reto, el proyecto Monasca [mon17] es utilizado como una herramienta de almacenamiento de métricas que permite el acceso eficiente a la información por medio de un API optimizado. Esta propuesta se presenta como solución al reto de escalabilidad [cei17b]. La Figura 13.10 muestra la relación existente entre Ceilosca, la arquitectura de monitorización SELFNET y el modo de operación.

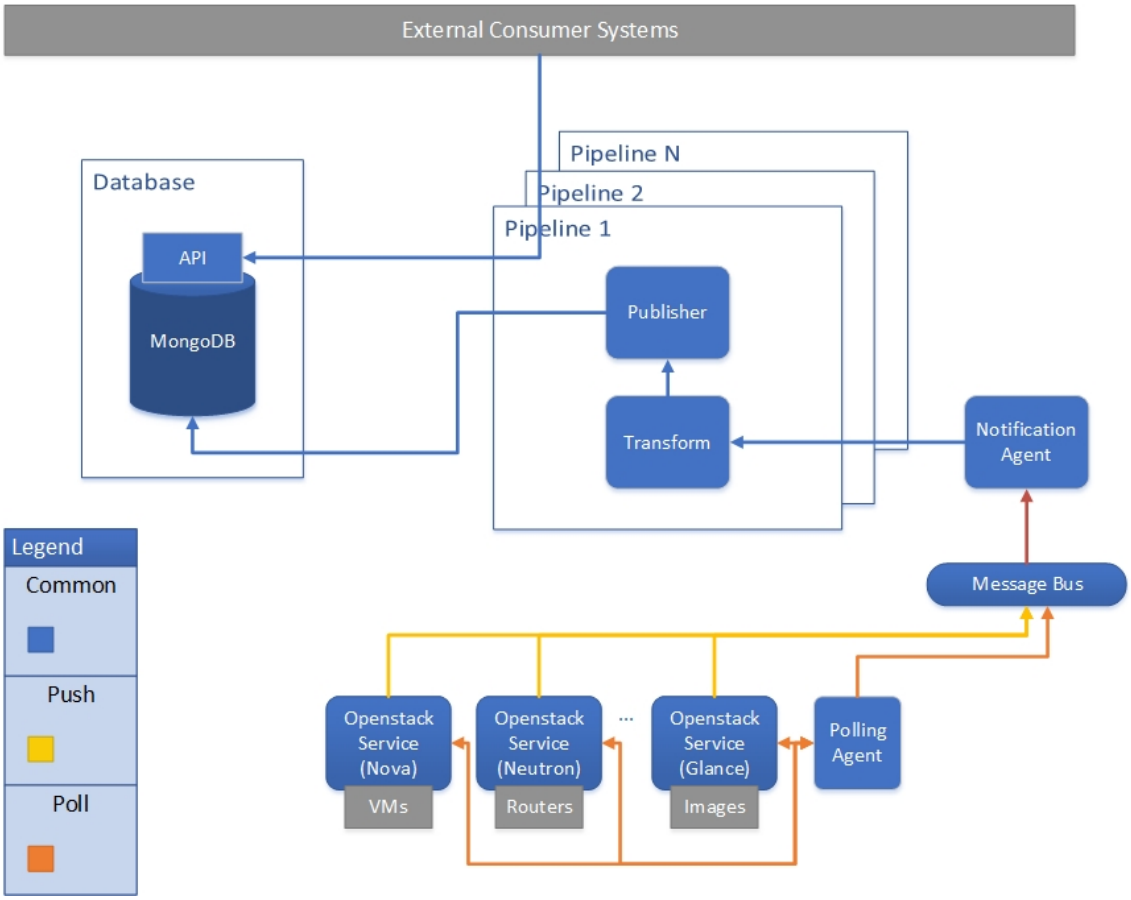


Figura 13.9: Arquitectura ceilometer

La arquitectura se compone de los siguientes elementos:

- **Fuentes de datos**. Este componente corresponde a las fuentes nativas de ceilometer (servicios OpenStack), así como nuevos plugins que permiten la recolección de métricas de sensores. La funcionalidad de este componente es explicado en la Sección 13.2.3.

- **Colector**. Este componente está compuesto por el agente de notificación, el canal de Ceilometer y el API de Monasca. Su función se encuentra detallado en la Sección 13.2.5.

- **Almacenamiento primario**. Este componente almacena métricas y eventos y es representada por la base de datos de Ceilosca. El componente es detallado en la Sección 13.2.6.

- **Interfaz Monitorización API**. Representa la interfaz expuesta por la arquitectura de monitorización. En la presente implementación se encuentra representada por la interfaz de Ceilometer.

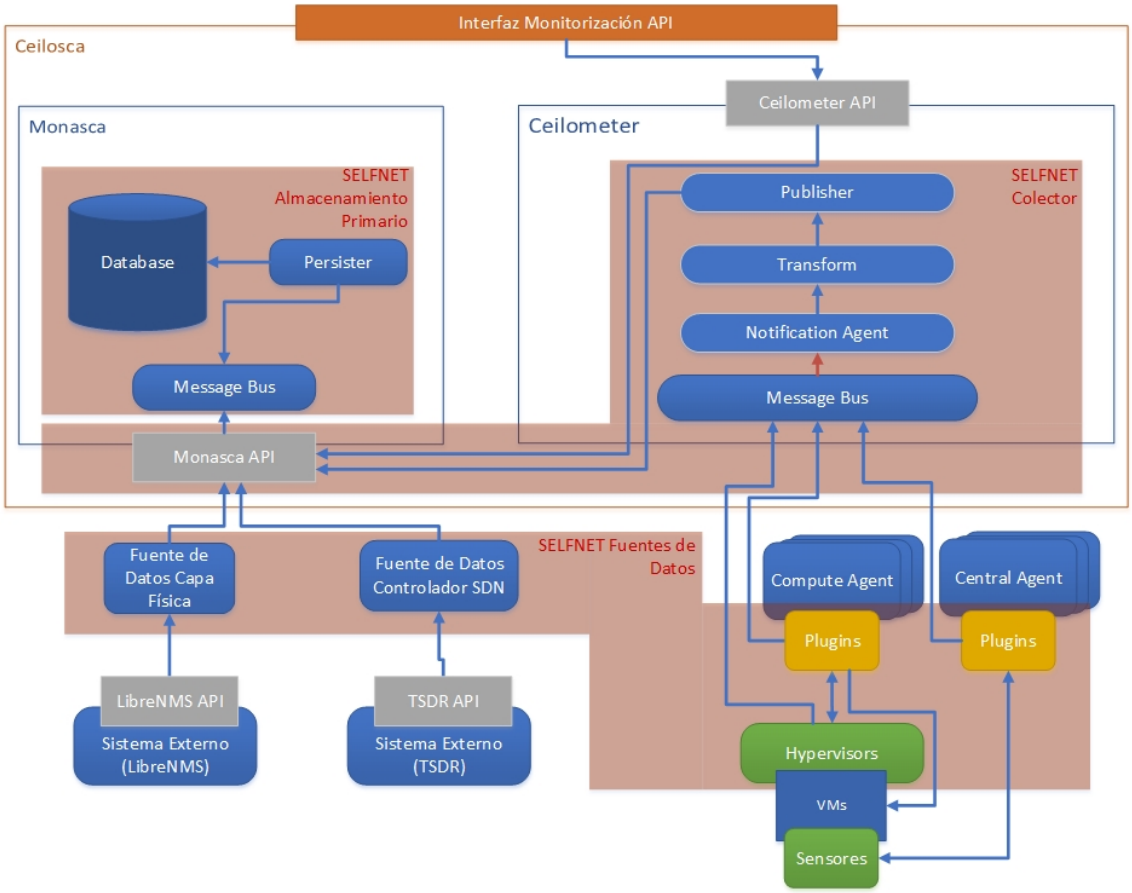


Figura 13.10: Relación entre ceilosca y la arquitectura SELFNET

13.4.2 Descriptores de Sensores Virtuales

Los descriptores de sensores virtuales son responsables de recibir, filtrar y almacenar la información correspondiente a los sensores disponibles en el catalogo SELFNET. Con este fin, el modulo se encuentra implementado por los siguientes componentes:

- **Cola de mensajes**. Se encarga de escuchar cualquier acción desplegada en el catálogo.
- **Base de datos local**. Almacena las características relevantes del sensor virtual, como por ejemplo el protocolo de comunicación.
- **Servidor REST**. Corresponde a la interfaz entre los descriptores y el gestor de fuentes de datos.

La arquitectura de monitorización escucha las notificaciones enviadas por la subcapa de integración VNF. La interfaz correspondiente utiliza RabbitMQ como bus de mensajes y es desarrollado usando Python3 y la arquitectura Web Falcon. Debido a que la información relacionada con el sensor no es relacionar y únicamente describe al sensor, el gestor de almacenamiento es MongoDB.

Los descriptores exponen una interfaz REST para que el gestor de fuentes de datos pueda solicitar información relativo a un sensor específico. En este caso, el cliente utilizará el tipo de sensor (sensor type) como identificador y recibirá como respuesta un JSON con la información disponible en el catálogo. La cola de mensajes cliente es preparado para recibir 3 tipos de acciones:

- **Crear**. Crea y añade nueva información de metadatos a la base de datos local.
- **Actualizar**. Reemplaza los metadatos con la información actual.
- **Eliminar**. Elimina la información de metadatos de un sensor específico.

Cada métrica recibida es compuesto por un único nombre "metric-name" y un conjunto de muestras, cada uno identificado por un nombre, un recurso, unidad y tipo (ej. acumulativo o delta).

13.4.3 Gestor de Fuentes de Datos

El gestor de fuentes de datos se encarga de la instanciación y configuración de las fuentes de datos. Asimismo, establece el canal de comunicación con el sensor para iniciar la recolección de información. Con el fin de recibir notificaciones sobre las acciones por el orquestador con respecto a un sensor determinado (instanciación, reconfiguración, eliminación), una interfaz basada en el bus RabbitMQ es implementado. De igual manera, la notificación recibida es clasificada por el componente de selección de acciones, tal como se muestra en la Figura 13.11. Este módulo tiene como función analizar el tipo de fuente instanciado, identificarlo y seleccionar la respectiva action que tomará el gestor.

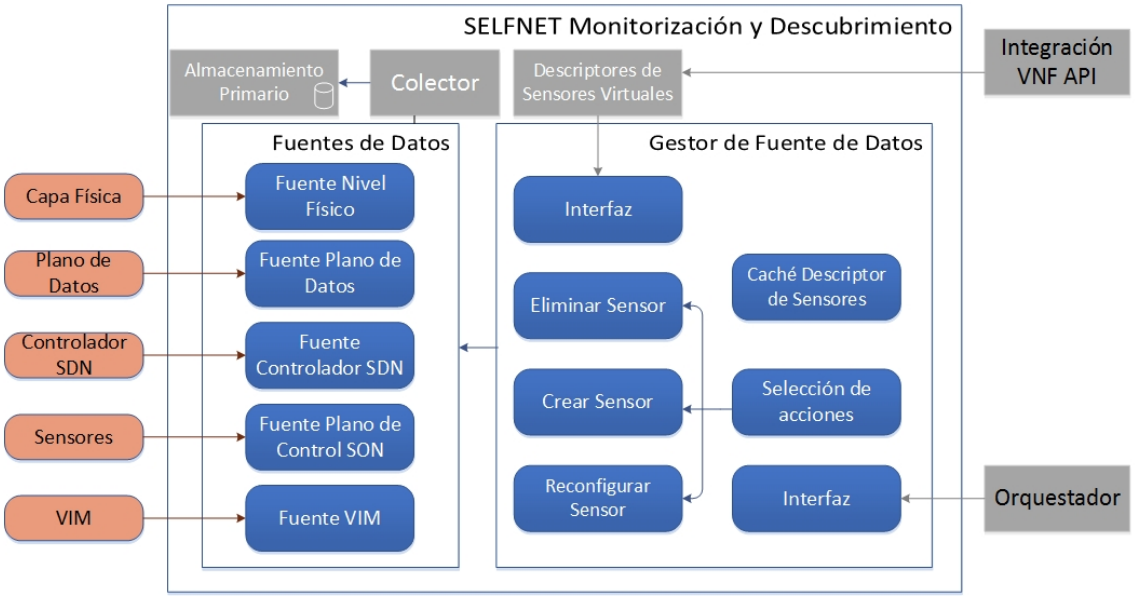


Figura 13.11: Gestor de fuentes de datos: operaciones del sensor

Como complemento al selector de acciones, 3 bloques funcionales han sido implementados: i) el gestor de nuevos sensores, ii) el gestor de eliminación de sensores

y iii) el gestor de cambios de sensores. Adicionalmente, una interfaz entre el gestor de fuentes de datos y los descriptores es implementada con el fin de recibir la información o metadatos de sensores.

13.4.4 Instancias de Fuentes de Datos

Una fuente de datos se encarga de enlazar el módulo de monitorización y el elemento encargado de obtener las métricas específicas del estado de la red. Con este objetivo, se han utilizado multiples herramientas de código abierto que permiten enlazar y obtener métricas a diferente nivel. En la Tabla 13.8 se resume la lista de herramientas utilizadas en la implementación.

Tabla 13.8: Herramientas utilizadas en la implementación de la fuente de datos

Fuente	Aplicación utilizada
Infraestructura Física	LibreNMS [lib17]
Controlador SDN	ODL [MVTG14]
Plano de Datos	ODL-TSDR [tsd17]
VIM	OpenStack [SAE12]
Sensores Plano Control	Ceilometer [cei17a]

13.4.5 Colector

Tomando en consideración la relación existente entre la arquitectura SELFNET y Ceilosca. El colector es compuesto por el Monasca API y 4 componentes de Ceilometer: “Message Bus”, “Notification Agent”, “Transform” y “Publisher”. El “Notification Agent” recibe las notificaciones que provienen del bus de mensajes. Asimismo, el “Transform” recibe dichas notificaciones y normaliza la información. Finalmente, el “Publisher” envía la información normalizada hacia un destino específico: base de datos de Ceilosca o consumidores externos. En la presente implementación, el destino principal es la interfaz que comunica con el agregador. Adicionalmente, la interfaz de Monasca puede ser utilizada para enviar información sin el uso de agentes Ceilometer, es decir de fuentes no pertenecientes a servicios OpenStack (ej., LibreNMS, ODL-TSDR).

13.4.6 Almacenamiento Primario

Ceilometer ofrece alternativas para almacenar métricas y eventos. Por defecto, Ceilometer almacena métricas y eventos en una base de datos tipo MongoDB, una base NoSQL. Actualmente, Ceilometer recomienda el cambio de MongoDB por el servicio TDBaaS (Time Series Database as a Service) para guardar métricas, mientras que los eventos mantienen MongoDB para casos de uso de baja latencia.

La principal razón para el cambio son los potenciales problemas de escalabilidad que MongoDB presenta en caso de múltiples solicitudes (el rendimiento recae con altas cantidades de información almacenada). La presente implementación de la arquitectura SELFNET brinda algunas alternativas para el almacenamiento de información. Por un

lado se utiliza Monasca para guardar datos usando InfluxDB (TDBaaS como Gnocchi). Por otro lado, Ceilosca habilita una interfaz con funcionalidades adicionales, tales como max, count, avg.

13.4.7 Validación - Interfaz Gráfica

La presente arquitectura de monitorización y descubrimiento ha sido implementado en un ambiente virtualizado como prueba de concepto. La implementación ha sido realizada sobre un servidor físico (ThinkServer TD350: Intel Xeon Serie E5-2600 v3 -16 cores, DDR4 512 GB - 2133 MHz). Se han desplegado 6 máquinas virtuales a través del hypervisor Virtualbox, tal como se describe en la Figura 13.12. Las VMs representan el *Controller Node, Compute Node, Telemetry Node, Opendaylight Node, LibreNMS Node* y *Monitoring Server*.

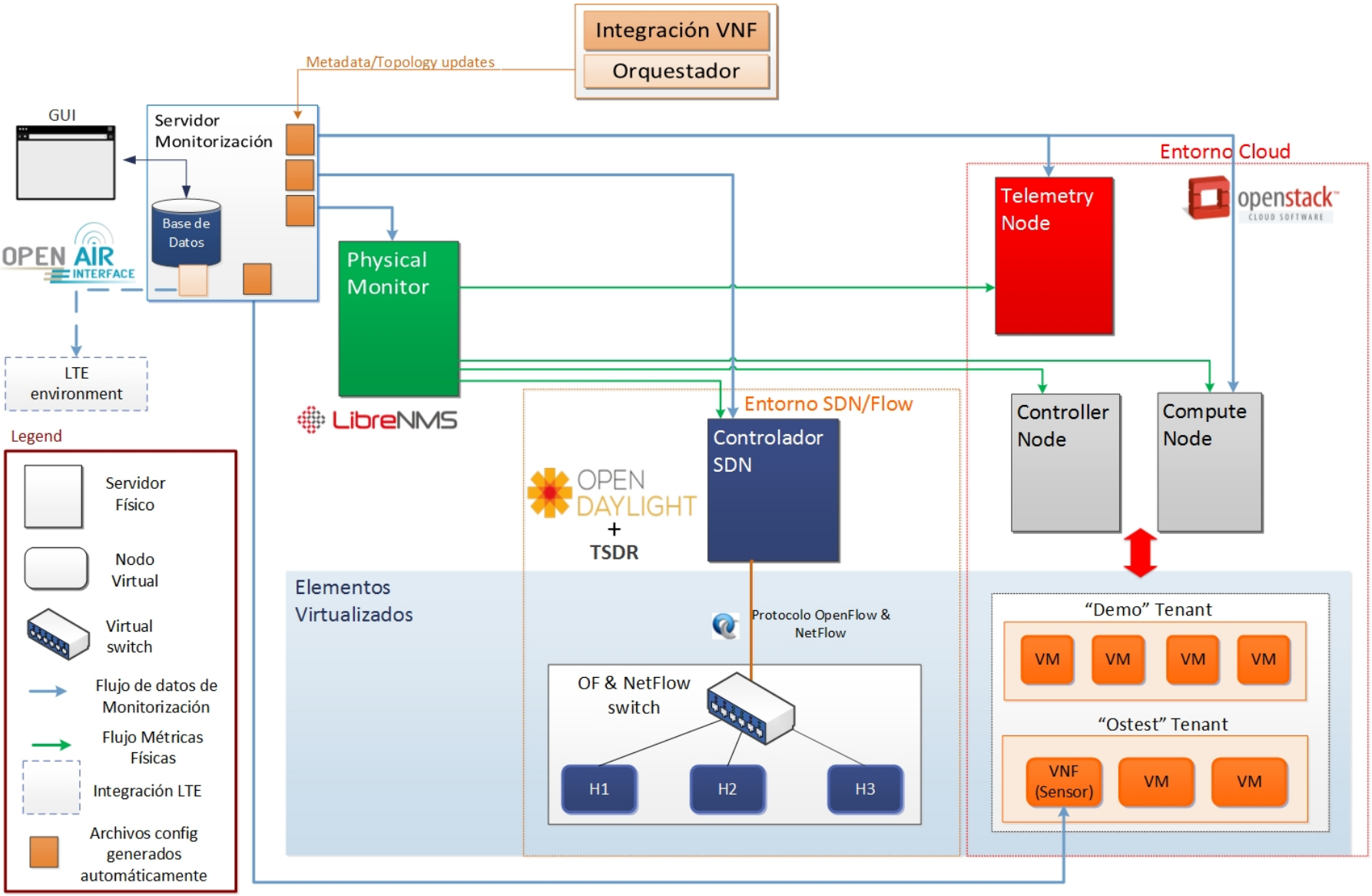


Figura 13.12: Prueba de concepto de la arquitectura SELFNET

El servidor de monitorización recolecta la información enviada por diferentes fuentes de datos: métricas físicas (LibreNMS), entorno cloud (OpenStack), entorno SDN/flow (ODL+TSDR), entorno LTE (OpenAir) y sensores. El element Physical Monitoring recolecta métricas correspondientes a los elementos físicos SDN Controller, Compute Node, Telemetry y Collector, los cuales emulan un entorno cloud. Por su parte, el Telemetry Node recolecta información de los servicios ejecutándose en OpenStack. Asimismo, el Monitoring Server recolecta la información de los sensores instanciados. La comunicación entre los diferentes nodos virtuales se los realiza mediante dos redes:

- **Red de Administración (MGNT)**. Se encarga de la comunicación entre OpenStack, LibreNMS y servicios ODL.
- **Red Pública**. Ofrece acceso a internet a los nodos que lo requieran.

Adicionalmente, dos servicios OpenStack están principalmente relacionados con la comunicación de instancias virtuales (VMs): i) El servicio Nova habilita la instanciación de nuevas máquinas virtuales y ii) El servicio Neutron es responsable de la gestión de redes virtuales.

Con el fin de visualizar las capacidades de la arquitectura, se ha desarrollado una interfaz gráfica GUI. Se ha implementado una arquitectura three-tier basado en un entorno cliente - servidor. Tal como se muestra en la Figura 13.13, la GUI muestra una muestra de los datos recolectados por la capa física, virtual, LTE, sensores, SDN controller y plano de datos.

Por ejemplo, en el caso de la capa física, se muestran todos los dispositivos que han sido descubiertos y sus parámetros relevantes: *device identification, hostname, location, port* y *MAC address*. Por su parte, la capa virtual muestra la lista de tenants (dominios) y sus características relevantes (*ID, name and description*), y la información relacionada con las instancias virtuales, tal como se muestra en la Figura 13.14.

Por su parte, la vista a nivel de flujos describe las métricas obtenidas por OpenFlow y Netflow. De igual manera, la vista de sensores (*Sensor View*) muestra todos los sensores que se encuentran desplegados en la infraestructura, sus parámetros relevantes: *sensor identification, IP Address, tenant identification, tenant network identification, location, type of sensor*. Adicionalmente se recogen las métricas y estadísticas recibidas, tal como se muestra en la Figura 13.15.

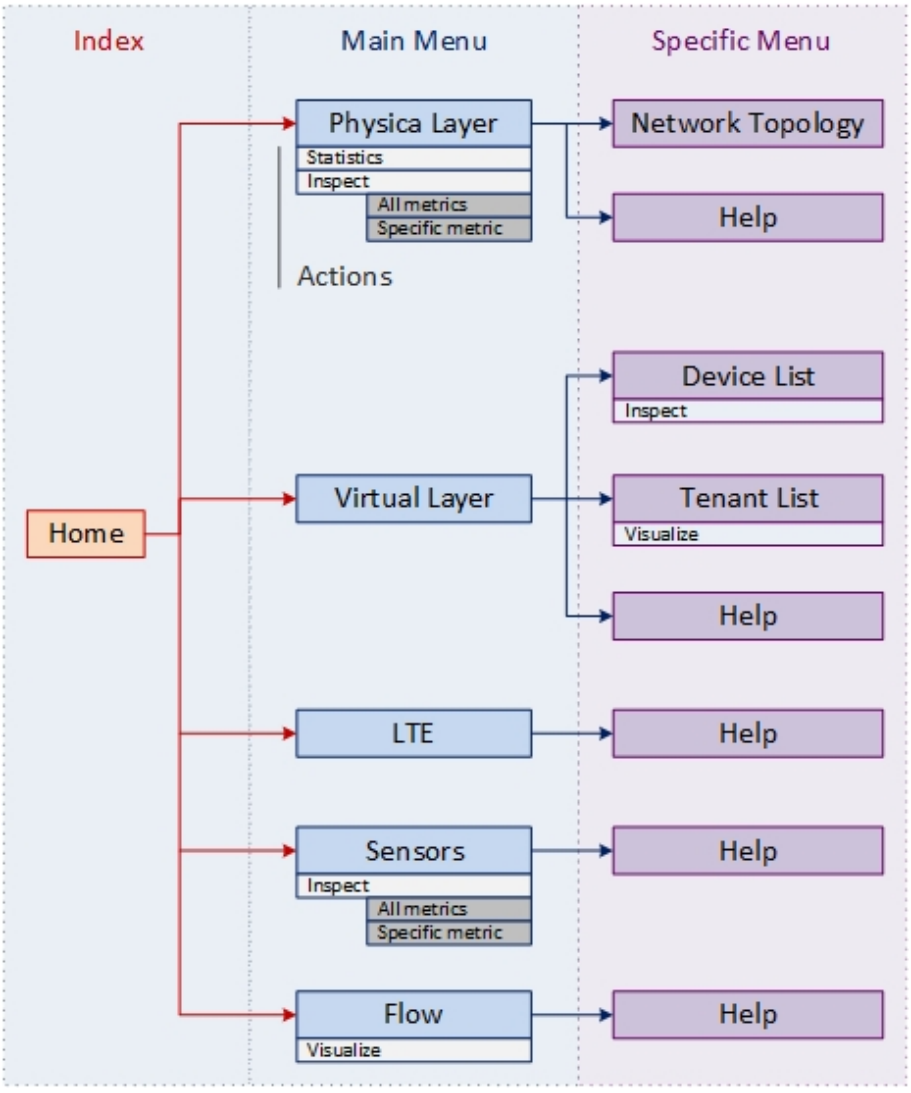
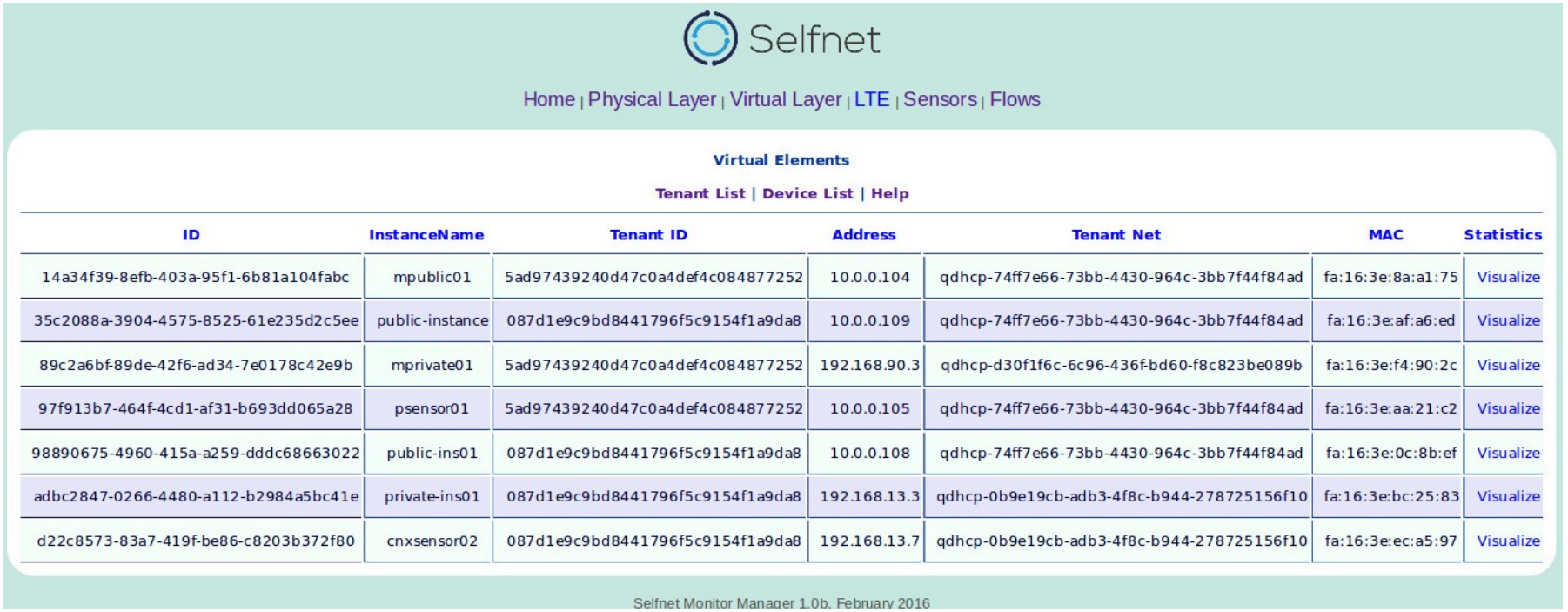


Figura 13.13: Interfaz gráfica de la arquitectura monitorización SELFNET



ID	InstanceName	Tenant ID	Address	Tenant Net	MAC	Statistics
14a34f39-8efb-403a-95f1-6b81a104fabc	mpublic01	5ad97439240d47c0a4def4c084877252	10.0.0.104	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:8a:a1:75	Visualize
35c2088a-3904-4575-8525-61e235d2c5ee	public-instance	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.109	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:af:a6:ed	Visualize
89c2a6bf-89de-42f6-ad34-7e0178c42e9b	mprivate01	5ad97439240d47c0a4def4c084877252	192.168.90.3	qdhcp-d30f1f6c-6c96-436f-bd60-f8c823be089b	fa:16:3e:f4:90:2c	Visualize
97f913b7-464f-4cd1-af31-b693dd065a28	psensor01	5ad97439240d47c0a4def4c084877252	10.0.0.105	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:aa:21:c2	Visualize
98890675-4960-415a-a259-dddc68663022	public-ins01	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.108	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:0c:8b:ef	Visualize
adbc2847-0266-4480-a112-b2984a5bc41e	private-ins01	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.3	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:bc:25:83	Visualize
d22c8573-83a7-419f-be86-c8203b372f80	cnxsensor02	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.7	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:ec:a5:97	Visualize

Figura 13.14: Vista de los elementos virtuales presentes en la infraestructura

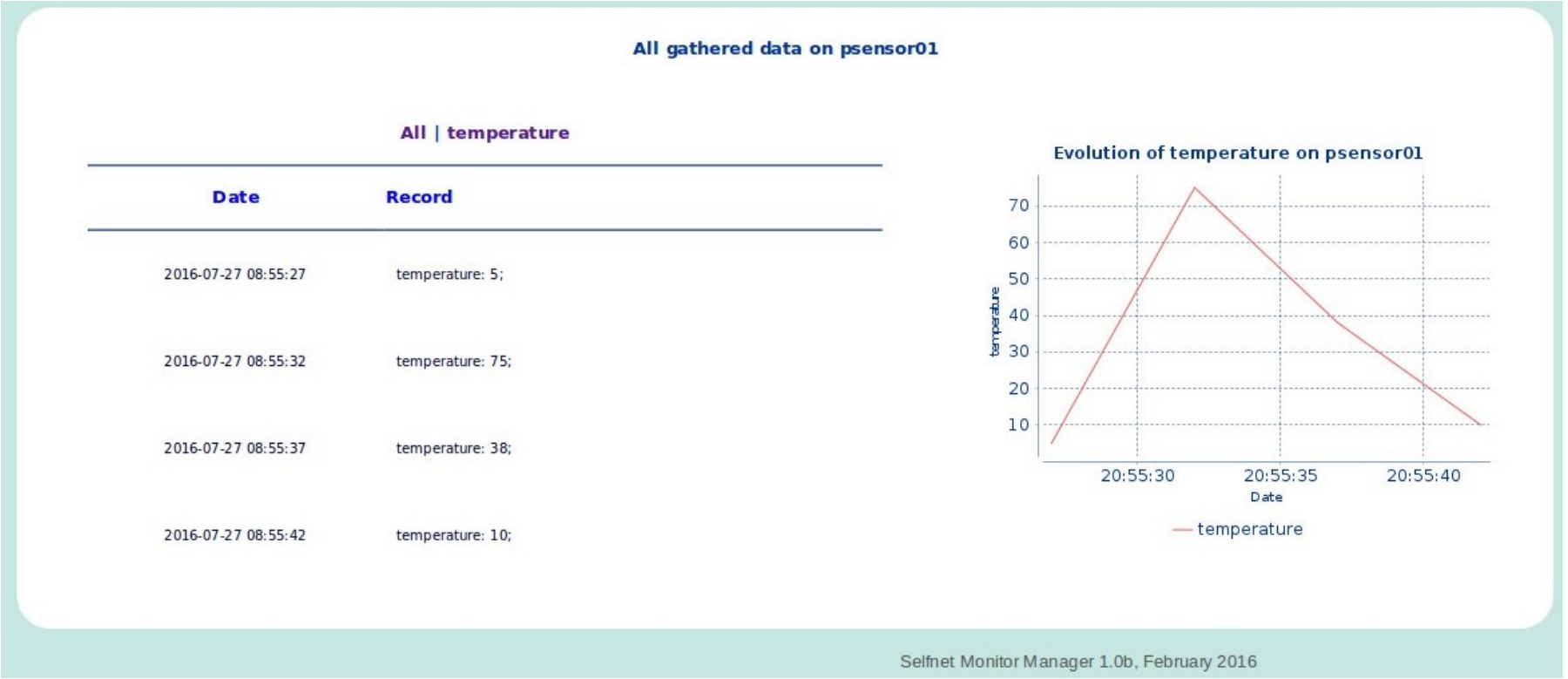


Figura 13.15: Detalles de las métricas recolectadas por un sensor SELFNET

Es importante destacar que en el presente ejemplo, el nombre “psensor01” corresponde a un sensor de temperatura. “psensor01” se encuentra ejecutándose en una instancia virtual en un tenant específico (ostest), y las métricas se recolectan cada 5 segundos. Las métricas obtenidas son definidas en la capa de registro (onboarding) y la información de instancia se reciben por medio del orquestador. El presente ejemplo muestra las métricas recibidas y su evolución a través del tiempo.

13.5 Resumen

El presente capítulo describe la arquitectura de monitorización SELFNET. De igual manera, los procedimientos llevados a cabo por la infraestructura y las tareas llevadas a cabo por los elementos internos son definidos. Finalmente, se explica la implementaicón y los detalles de la correspondiente interfaz gráfica.

Capítulo 14

Conclusiones y Trabajo Futuro

Los resultados de la presente Tesis se enmarcan en el estudio de las Redes Definidas por Software SDN y Virtualización de Funciones de Red SDN en términos de Monitorización, Calidad de Servicio y Gestión de Redes. De igual manera, la investigación se motiva en un análisis práctico de los requerimientos, visión e indicadores KPI de las arquitecturas 5G.

Tal como se describe en las primeras secciones del presente documento, uno de los objetivos de la presente investigación es el estudio del estado del arte de los temas SDN, NFV y de los retos actuales en la mejora de los servicios de red. De igual manera, se han analizado las redes 5G, especialmente en su propuesta de un entorno de gestión autónomo de recursos de red, computacionales y almacenamiento. La visión de reemplazar la actual configuración manual de equipos por sistemas autónomos de gestión ha abierto un nuevo mercado basado en la personalización de servicios de red. En este contexto, la presente Tesis es parte del proyecto SELFNET, el cual tiene como misión el diseño e implementación de un sistema autónomo de gestión de red para arquitecturas 5G. SELFNET involucra la gestión autónoma de tareas fundamentales, tales como automatización de monitorización, tareas de mantenimiento, provisión de servicios, detección de amenazas, etc.

Con el advenimiento de NFV como un complemento necesario para SDN y dado que habilita la personalización de servicios de red, ambas tecnologías son la base que utiliza SELFNET para el desarrollo de la arquitectura de gestión autónoma. SELFNET integra el paradigma de gestión autónoma con el uso de minería de datos, algoritmos de aprendizaje, reconocimiento de patrones para identificar el estado de la red y automáticamente tomar decisiones para reducir el riesgo de fallos o degradación de la calidad de servicio. En este sentido, la arquitectura permite la gestión autónoma de funciones de red virtuales que pueden ser utilizados para resolver problemas de red o mejorar la calidad de servicio. La arquitectura SELFNET sigue estrictamente un enfoque por capas. Las capas y subcapas han sido definidos en concordancia con los actuales estándares para SDN, como los promulgados por el Open Networking Foundation, y NFV como el ETSI VNF. Adicionalmente, SELFNET combina estos estándares y los adapta para arquitecturas 5G.

Dentro de los principales objetivos propuestos por SELFNET, la monitorización y almacenamiento de información relevante sobre el estado de la red es un reto clave. El proyecto SELFNET monitoriza el estado de la red y el rendimiento del sistema que serán utilizados para brindar inteligencia y capacidades avanzadas de gestión. El presente trabajo

de investigación propone la arquitectura de monitorización SELFNET, el cual permite unificar la recolección de múltiples fuentes. La arquitectura se centra en obtención y almacenamiento de métricas correspondientes a equipos físicos, equipos virtuales, entornos en la nube, métricas a nivel de flujo, tráfico SDN e información enviada por sensores.

El presente trabajo senta las bases para ofrecer un modelo abierto y personalizable para recolección de métricas de bajo nivel. Como resultado, la arquitectura prupuesta facilita el proceso de obtención de información y el tratamiento de altas cantidades de información de fuentes heterogéneas. La información almacenada se encuentra disponible para posteriores tareas de agregación, correlación y análisis.

Adicionalmente, el diseño y especificación de la arquitectura es implementada usando herramientas de código libre. Diferentes herramientas de monitorización han sido revisados y evaluados, y las soluciones más apropiadas han sido utilizadas y mejoradas para cumplimiento de los requerimientos. En particular, LibreNMS, Opendayligh TSDR, OpenStack Telemetry (Ceilometer) y Ceilosca han sido seleccionados como punto inicial para implementación de la arquitectura.

En concordancia con los objetivos propuestos por SELFNET en términos de mejorar las funcionalidades de monitorización, el presente trabajo de investigación también desarrolló arquitecturas de monitorización de Redes SDN que busca reducir la carga en CPU del controlador y los recursos de hardware manteniendo altos niveles de precisión. La arquitectura propone un módulo de orquestación junto a un método adaptativo de muestreo de la información basado en la carga del controlador y tamaño de la red. La implementación de la arquitectura en un controlador OpenFlow y las pruebas realizadas con un emulador de red y servidor de video demuestran la factibilidad de la arquitectura.

De igual manera, la presente tesis presenta una arquitectura SDN que ofrece mejoras en la calidad de servicio para diferentes servicios multimedia. La arquitectura usa OpenFlow, Virtualización y establece módulos e interfaces necesarios para integrar deferentes algoritmos de encaminamiento. Las ventajas de la arquitectura son probados con la implementación de una función de rendimiento de red y algoritmo de encaminamiento. Los experimentos con información de video streaming han comprobado la mejora en la calidad de video (PSNR, SSIM, MOS) en comparación con la estrategia del mejor esfuerzo.

14.1 Trabajos Futuros

Considerando que las tecnologías SDN, NFV y autogestión de infraestructuras móviles se encuentran en una fase temprana de desarrollo, la presente investigación resalta algunos trabajos que serán encaminados en el futuro.

En relación con la gestión autónoma de redes SDN/NFV, el proyecto SELFNET presenta retos importantes a ser analizados. Las métricas de bajo nivel que se encuentran disponibles deben tener un tratamiento adicional con procesos agregación y correlación. En este sentido, es necesaria la coordinación y orquestación de todas las reglas solicitadas por los proveedores de servicio y diseñadores de los casos de uso. La definición de las métricas de alto nivel que sinteticen el comportamiento de la red es un tema abierto de discusión. De igual manera, la posibilidad de usar métricas de red de alto nivel que permitan detectar

patrones y predecir el estado futuro de la red es una tarea de amplio análisis en el futuro.

En este sentido, es preciso un modelo de evaluación de la situación de la red SA, el cual tenga en cuenta las operaciones de gestión tradicionales y el dinamismo esperado en las redes móviles 5G. En otras palabras, es importante el establecimiento y formalización del proceso de análisis de la información enviada por la arquitectura SELFNET de monitorización y descubrimiento. El diseño del modelo de análisis tiene que igualmente habilitar la personalización de parámetros, funciones de análisis y reglas de diagnóstico. Asimismo, el modelo debe tener la capacidad de inferir el estado de la red y predecir potenciales amenazas. Por lo tanto, la inclusión de capacidades de diagnóstico y predicción en el contexto de las redes 5G facilitará el proceso de toma de decisiones y gestión de servicios y aplicaciones por medio de acciones reactivas y proactivas.

En relación a la monitorización de redes SDN, los retos futuros incluyen la mejora de los diferentes algoritmos de monitorización, analizando nuevas métricas del plano de datos, tales como tiempo de establecimiento de conección, jitter o duplicación de paquetes. De igual manera, es importante la coordinación entre plano de datos y de control que evite el desbalance en la carga entre ambas capas. El uso de equipos separados para mininet y floodlight mejorarán la calidad de los resultados. Así mismo, la coexistencia entre plataformas tradicionales y redes SDN es un desafio abierto para la comunidad científica.

En relación a la optimización de la calidad de servicio en redes SDN, los retos incluyen el mejoramiento de algoritmos de calidad de servicio y rendimiento de la red. De igual manera, se requiere el desarrollo de experimentación con enlaces de alta velocidad, grandes cantidades de flujos de datos y bajas tasas de pérdidas y diferentes tipos de controladores SDN. Características de optimización adicionales, tales como encaminamiento multicamino, tunelización y gestión de recursos en función de prioridades también pueden ser incluidos en futuras propuestas.

Part IV

Papers Related to This Thesis

Appendix A

List of Papers

1. A. L. Valdivieso Caraguay,L. I. Barona López, L. J. García Villalba: Evolution and Challenges of Software Defined Networking. Workshop on Software Defined Networks for Future Networks and Services (SDN4FNS), Trento-Italy, November 12 - 13. 2013.

2. A. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, L. J. García Villalba: SDN: Evolution and Opportunities in the Development IoT Applications. International Journal of Distributed Sensor Networks, Hindawi Publishing Corporation, 10(5), pp. 1-10, May 2014

3. A. L. Valdivieso Caraguay, J. A. Puente Fernández, L. J. García Villalba: Framework for Optimized Multimedia Routing over Software Defined Networks, Computer Networks, 92(2), 369-379, December 2015

4. A. L. Valdivieso Caraguay, L. I. Barona López, L. J. García Villalba: An Overview of Integration of Mobile Infrastructure with SDN/NFV Networks. In The 7th International Conference on Information Technology, Amman, Jordan, May 12 - 15, 2015.

5. A. L. Valdivieso Caraguay, J. A. Puente Fernández, L. J. García Villalba: An Optimization Framework for Monitoring of SDN/OpenFlow Networks, International Journal of Ad Hoc and Ubiquitous Computing IJAHUC, (Accepted September 2015, In Press).

6. A. L. Valdivieso Caraguay, L. J. García Villalba, et al: Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks, Sensors, 17(4), pp. 1-31, March 2017.

Additional Publications

1. J. P. Santos, R. Alheiro, L. I. Barona López, L. J. García Villalba, et al: SELFNET Framework Self-healing Capabilities for 5G Mobile Networks. Transactions on Emerging Telecommunications Technologies, 27(9): 1225-1232, June 2016.

2. P. Neves, A. L. Valdivieso Caraguay, L. J. García Villalba, et al: Future Mode of Operations for 5G - The SELFNET Approach Enabled by SDN/NFV. Computer Standards & Interfaces. (Accepted on December 2016, In Press)

3. L. J. García Villalba, A. L. Valdivieso Caraguay. Use Cases Definition and Requirements of the System and its Components. SELFNET Project, October 2015. $https://doi.org/10.18153/SLF-671672-D2\_1$.

4. L. J. García Villalba, A. L. Valdivieso Caraguay. Report and Prototypical Implementation of the Monitoring and Discovery Module. SELFNET Project, September 2016.

SDN4FNS 2013

2013 Workshop on Software Defined Networks for Future Networks and Services

November 11-13, 2013
Trento, ITALY

IEEE
Future Directions Committee

IEEE COMMUNICATIONS SOCIETY

IEEE computer society

WIRELESS WORLD RESEARCH FORUM

IEEE Catalog number: CFP13SDN-ART ISBN: 978-1-4799-2781-4

Evolution and Challenges of Software Defined Networking

Ángel Leonardo Valdivieso Caraguay, Lorena Isabel Barona López,
Luis Javier García Villalba, *Senior Member, IEEE*

***Abstract*—Software Defined Networking (SDN) proposes the separation of the control plane from the data plane in network nodes. Furthermore, Openflow architecture through a centralized control of the packet forwarding engines enables the network administrators to literally program the network behavior. The research and results of experiments show clear advantages over traditional network architectures. However, there are open questions to be solved in order to integrate SDN infrastructure and applications in production networks. This paper presents an analysis of the evolution of SDN in recent years. Additionally, this piece of work also describes some interesting SDN/Openflow research initiatives and applications. Finally, there is a disscussion on the main challenges of this new technology.**

***Index Terms*—Future Internet, Software Defined Networking, OpenFlow.**

I. Introduction

Network technologies have become in a critical element in almost all human activity. The number of devices as well as the amount of traffic moving in Internet is growing exponentially. However, the development of new protocols and services (*mobility, VoIP, QoS, video streaming*) has been limited by the hard union between specialized packet forwarding hardware and operating systems running hundreds of static protocols (some of them private) and only allowing the network administrator only the configuration of the network equipment. Additionally, the deployment time of a new idea from the design, simulation, testing, publication in a standard and finally the installation in network equipment can take some years. Clearly, the new generation networks need to change the rigidity of actual network architectures.

Software Defined Networking (SDN) is an architectural innovation that proposes the separation of *control plane* and the *data plane* enabling their independent evolution and development. Ethane [1] was one of the first SDN initiatives. This model was proposed for enterprise networks and uses a centralized control architecture. However, this implementation required the deployment of custom switches (OpenWrt, NetFPGA, Linux) to support Ethane Protocol.

Today, the principal crystallizing of SDN is Openflow [2], an open architecture initially developed as a way of allowing researchers to run experiments on heterogeneous networks without affecting the real users traffic. Openflow [2] aims at providing more configuration options in the switch dataplane.

The Openflow specification [3] establishes the rules of communication between *data plane* and a centralized *control plane* and allow the entire network to be controlled through users software applications (APIs). The Open Networking Foundation ONF [4] brings together about 90 companies and is dedicated to releasing, promoting and adopting of OpenFlow specification [3].

This implementation of Openflow [3] removes the limitation of rigidity of static protocols, opens the possibility of rapid innovation and led research community to develop new paradigms in network technologies. Some examples of this evolution are: Virtualization [5], [6], High Level Network Programming Languages [7], [8], optimization of Quality of Experience QoE applications [9], [10]. However, new challenges and unanswered questions appear when trying to implement these ideas in production networks: Modeling and Performance [11], Resilience and Recovery [12], Security [13], Convergence and Management [14], [15]. This paper presents an analysis of SDN/Openflow [2] technologies and discusses the challenges and open question in order to implement them in production networks.

The rest of the paper is outlined as follows: Section II defines the Openflow architecture. Then, Section III presents the actual research and experimentation. Next, Section IV analyzes the SDN challenges. Finally, Section V concludes with the discussion and conclusions.

II. OPENFLOW

The Openflow architecture [2] consists of three principal elements: an Openflow switch, an external controller and the Openflow Protocol [3] which establishes the communication between switch and controller through a secure channel.

An Openflow switch consist of one or more *flow tables* and a *group table* used for packet lookup and forwarding. Each *flow table* is composed of a set of match fields, counters and instructions. The packet fields to be compared with the match fields can be indistinct of the second, third or fourth layer in TCP /IP architecture. The number of the packet fields to be matched depends on the version of the Openflow protocol [4], [16]. The recent version of Openflow is v1.3 and specifies a number of 40 match fields including support to IPv6. However, the most widely used version is v1.0 with a number of 12 match fields.

The authors are with the Group of Analysis, Security and Systems (GASS), http://gass.ucm.es/en, Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases s/n, Ciudad Universitaria, 28040 Madrid, Spain (e-mail: {angevald, lorebaro, javierv}@ucm.es).

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

If the header of an incoming packet is similar to a match field in a *flowtable*, the corresponding actions are taken. The Openflow Protocol [3], [4] specifies the set of mandatory and optional actions that a switch can take. The principal mandatory actions are: send the packet by a given port, send packet to the controller or drop the packet. In the case that incoming packet doesnt match with a match field, the switch can be configured to send it to the external controller or drop the packet. The external controller has the responsibility of analyzing the information received by switches and controlling their actions through the maintenance of the *flow tables* in switches.

III. RESEARCH AND EXPERIMENTATION

The direct manipulation of the forwarding plane on network devices allows researchers the possibility of designing new services, routing protocols, security models, and directly testing their functionality directly in the network. Next, there is a revision of important SDN research initiatives.

A. Virtualization

Similar to computer virtualization layer, where a hardware abstraction permits slicing and sharing the hardware resources with different Operating Systems OS in a host, the goal of network virtualization is to isolate multiple logical networks, each of them with completely different addressing and forwarding mechanism, but sharing the same physical infrastructure. SDN provides the opportunity to reach a network hardware abstraction layer.

Flowvisor [5] is a virtualization platform that uses Openflow [2] to sit logically between control and forwarding paths. Flowvisor [5] acts as a transparent proxy between controllers and switches. Then, it creates strong and transparent *virtual slices* according to administrators established policies and ensures isolation in terms of bandwidth, flowspace and switch CPU load. The user can only observe and control only their own slice. Additionally, it is possible to slice a *virtual slice* and create like hierarchies of virtualized networks (Fig.1).

In [6] there is a demonstration of four successful networking experiments (load balancer, wireless streaming video, traffic engineering and hardware prototyping experiment) using Flowvisor [5], each with its own slice. However, the experiments show some problems to be solved: the unexpected interaction with other deployed network devices, increase of broadcast traffic emitted from non-OpenFlow devices and some violations of the CPU isolation, especially when one slice inserts a forwarding rule that is handled via the switches slow path.

Another important aspect is the integration between network operations and computer virtualization. In computer virtualization, the different virtual machines VMs require a network access layer that provides inter- and intra-VM connectivity and provide similar network functions to the physical layer. The common model is to establish communication between virtual nodes and physical NIC implementing typical L2 switching or L3 routing. This makes the network management of virtual environments difficult, for example the migration of VMs between different physical servers. In this approach, SDN and network virtualization can help to achieve these goals.

Open vSwitch [17] is a software switch built for virtual environments. This switch exports an interface for fine-grained network control. Additionally, it has a logical partition of the forwarding plane through a flexible, table-based forwarding engine. The forwarding plane has an external interface and can be managed for example by Openflow [2]. With this abstraction, the controller can obtain a logical view of multiple Open vSwitches running on separate physical servers.

Another interesting application of virtualization is the *virtual network migration (VNM)*. In typical networks, the migration or change of a network node require the re-configuration and the re-synchronization of the routing protocol. This causes high delays and packet loss. Consequently, the use of virtual nodes can significantly reduce the downtime.

In the VNM system proposed in [18], the SDN controller creates new flow entries for the new switch and redirects the paths from the initial to the new node. Then, the controller deletes the flow entries of the old switch making it feasible to be removed with security. The results of experiments show a total migration time of 5ms without packet loss. Moreover, the system could be dynamically reconfigured to place virtual networks on different physical nodes according to the time of day or the traffic demand to save energy (green networks).

B. Simulation and Testbeds

The possibility of debugging and testing new designs in a real environment is difficult and expensive. The network simulation is the first step to evaluating the idea previous implementation in a real production network. However, the current network simulators are expensive and dont offer support to Openflow [2].

Mininet [19] is the first open source simulator for Openflow networks. It uses OS-level virtualization features, including processes and network namespaces and allows a scale to hundreds of nodes (switches, hosts, and controllers) in a simple laptop. The user can run commands on hosts, verify switch operation and even induce failures or adjust link connectivity. Another advantage is that the code used to program a controller in the simulator is the same as the one to be deployed into a real network. However, Mininet [19] presents some limitations. The performance is directly related to the number and topology of virtual nodes and the available resources in the host. For this reason, the results of experiments in terms of bandwidth or time can vary from one computer to another.

In case the industry or research community may need to test experiments in a scaled real network infrastructure, there is worldwide in development different large scale testbeds. Global Environment for Network Innovations GENI [20] is a research infrastructure sponsored by the US National Science Foundation. Currently GENI [20] is deploying a SDN/Openflow [2] technology at their infrastructure (8 campuses interconnected).

The project OpenFlow in Europe: Linking Infrastructure and Applications OFELIA [21] bring together different European research institutes within the European Commisions FP7 ICT Work Programme. This project interconnects an infrastructure of 8 Openflow [2] islands allowing experimentation on multi-layer and multi-technology networks and providing realistic test scenarios and diverse and scalable resources. The Extending and Deploying Ofelia in BRAzil EDOBRA proposal [22] has the purpose of extending the OFELIA network to Brazil as a new OFELIA island.

C. High Level Network Policy Languages

In computer systems, the Operating-Systems (OS) facilitate the development of programs through high level abstractions of hardware and information resources. In the SDN paradigm, the entire behavior of the network is managed by the control layer. In the Openflow architecture [2], the control layer is materialized in a centralized controller. Clearly, the controller needs a Network Operating System NOS that facilitates the user create software programs to control the behavior of the network. Currently, there are several NOS. The most used open source NOS are: NOX/POX [23], Maestro [24], Beacon [25], Floodlight [26] each with its particular characteristics. NOX and POX are similar but implemented in C++ and Python respectively. Additionally, they use a group of application programming interfaces (APIs) to communicate and interact with switches. Beacon [25] and Floodlight [26] are NOS based on Java platform. Maestro [24] use multithreading looking for better performance and scalability. In [27] these kinds of controllers are denominated as *southbound*, because they interact directly with the forwarding layer and manipulate the state of individual devices. However, programming High Level operational tasks using this kind of NOS is still difficult. The user need to mix different match fields, actions, prorities for each switch looking for the espected behavior of the network. Moreover, the synchronization between control and data messages requires high attention and complicates the creation of new applications. For this reason, the *northbound* interfaces automatically translates the network policies (policy layer) in coordinated rules to the switches and ensure a correct behaviour of the network.

Procera [8] is a control architecture that includes a declarative policy language based on the notion of *Functional Reactive Programing (FRP)* [28] which provides a declarative, expressive and compositional framework for describing reactive and temporal behaviors according to network conditions. It also has event comprehensions to manipulate event streams and signal functions of windowing and aggregation. In [27], [29] uCap project is presented using Procera [8]. uCap [29] enables home users to monitor and manage end-host devices data usage in their home network. When the device reaches the data capacity allocated by the administrator, the system can disable the connection of this device. This system is useful especially for subscribers of ISPs that enforce monthly bandwidth caps.

Frenetic [7] is a network programming language that comprises two integrated sublanguages: a *Declarative Network Query Language* and a *Network Policy Management Library*. The *Network Query Language* reads the state of the network by installing low-level rules on switches that are transparent to the programmer. The *Network Policy Management Library* manages the policy of the forwarding packets using a functional, reactive programming based on FRP [28]. A working prototype of Frenetic is available in [30].

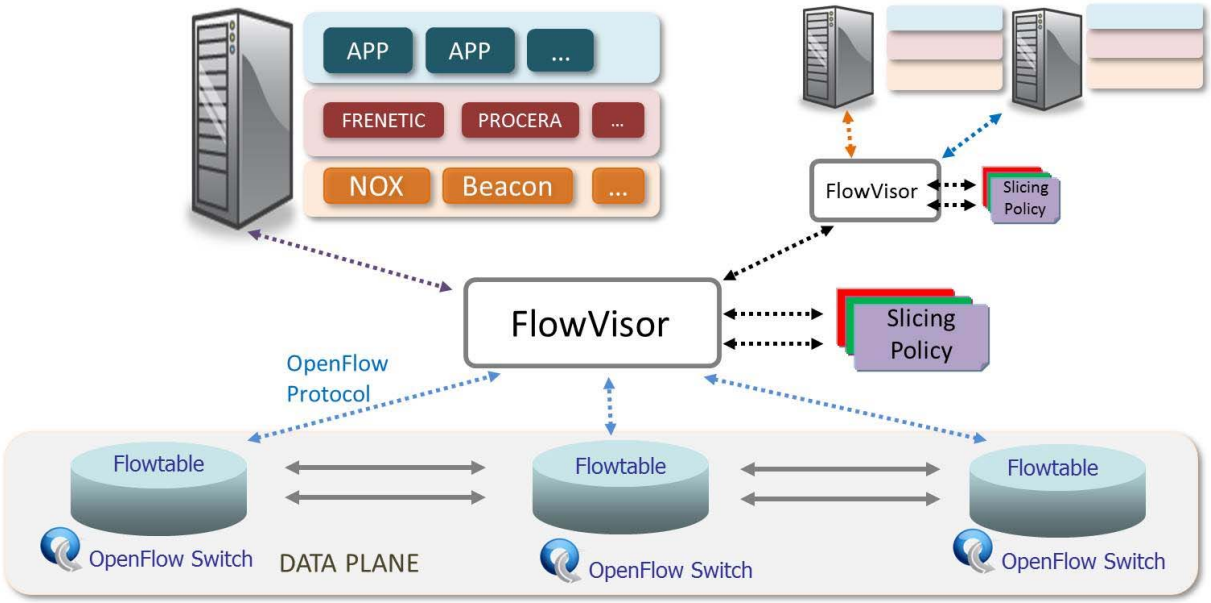


Fig. 1. Openflow Protocol, Virtualization and Network Operating Systems NOS

Another High Level Policy Language is Pyretic [31], which is designed to support modular programming. In other words, the programmer can write modules independently without interferance between them. Pyretic [31] establish different policy composition operators, including *parallel composition* and *sequential composition*. In *parallel composition* the policies are executed on the same packet and the results are combined. For example, send packet with header A to port 1 and packet with header B to port 2. In *sequential composition* the result of a policy A is the entry of the next policy B, such as the combination of balancing and switching policies (change the packet header and then routing based on the new header). The Fig.1 describes the location and function of the different abstraction levels (Openflow [2], Virtualization and NOS) within SDN paradigm.

D. Multimedia - Quality of Experience (QoE)

In multimedia systems, the notion of Quality of Experience QoE has growing since Quality of Service (QoS) is not powerful enough to express all features involved in a communication service. However, the term QoE doesnt yet have a solid, theoretical and practical concept.

The European Network on Quality of Experience in Multimedia Systems and Services (QUALINET [9]) emphasizes a working definition: *QoE is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and / or enjoyment of the application or service in the light of the users personality and current state*. Regardless, the use of this term QoE is increasing in the research community.

In [10], a SDN path optimization system for multimedia flows at the infrastructure layer improves QoE experience. The architecture consists of two main functions: QoS Matching and Optimization Function (QMOF) and the Path Assignment Function(PAF). QMOF reads the user multimedia parameters, determines whats feasible and looks for an optimal service configuration, while PAF maintains a network topology database. If the quality of the network is degraded, the system can react and dynamically modify the path parameters of the network prioritizing the users configuration.

Another important goal in multimedia communications and QoE is the capability to recovery a network path when a link fails (eg. *audio/video streaming*). In [32], a Openflow [2] based system is capable of recovering from a link failure using an alternative path. The system uses two well-known mechanisms: *restoration* and *protection*. In *restoration* method, when a path failure signal is received, the controller calculates and establishes an alternative path. In *protection* method, the controller anticipates a failure and calculates two disjoint paths (working and protected) before a failure occurs. The experiments show a recovery time of less than 50 ms for carrier-grade network.

IV. CHALLENGES

The deployment and experimentation of SDN/Openflow [2] initiatives have some important challenges at the moment to be successfully adopted at production networks. Below, there is a discussion of the most important aspects to considerate and the first proposed solutions.

A. Modeling and Performance

The implementation of SDN/Openflow requires the estimation of the number of controllers needed by a determinate topology and the localization of these controllers. This is a difficult question to answer. SDN establishes the separation of control and data planes, but doesnt necessarily order the existence of a single controller. SDN control plane may have single controller, a set of controllers in a hierarchy topology, or even any dynamic controllers topology. In [33], there is a complete study about this issue. The authors conclude that number and position of controllers depends on desired reaction bounds, metric choice and the network topology itself. The experiments show a decreased performance from each added controller, along with tradeoffs between metrics. However, the latency of a node connected with a single controller for medium sized network is similar to the response-time in actual production networks.

Another important factor is the selection of an appropriate NOS and the measure of its performance. In other words, how fast the controller responds to datapath request and how many datapath requests the controller can handle per second. However, it is also necessary to establish a balance between high performance and the productivity of developers (a developer friendly language). The experimentation [25], [34] show that the performance of the controller depends directly of programming language (C++, Java, Python) and the development environment.

The NOS previously named [23]–[26] are still in development and their features are constantly improved. For example, Fig. 2 and 3 show an evaluation of recently new Beacon improvements of performance in relation to other controllers. Tests were run on Amazons Elastic Computer Cloud using a Cluster Compute Eight Extra Large instance, containing 16 physical cores from 2 x Intel Xeon E5-2670 processors, 60.5GB of RAM, using a 64-bit Ubuntu 11.10 VM image (ami-4583572c) [25]. The experiment uses Cbench [35] utility for testing the throughput (number of transactions per second) and latency (time required for a packet to arrive at its destination through the network) of Openflow [2] controllers using a single thread.

Fig.2 shows the capacity of the different NOS handling constantly Packet In messages from 64 emulated switches as soon as possible. Fig.3 measures the average time between sending a single packet of an emulated switch and the reply from the controller.

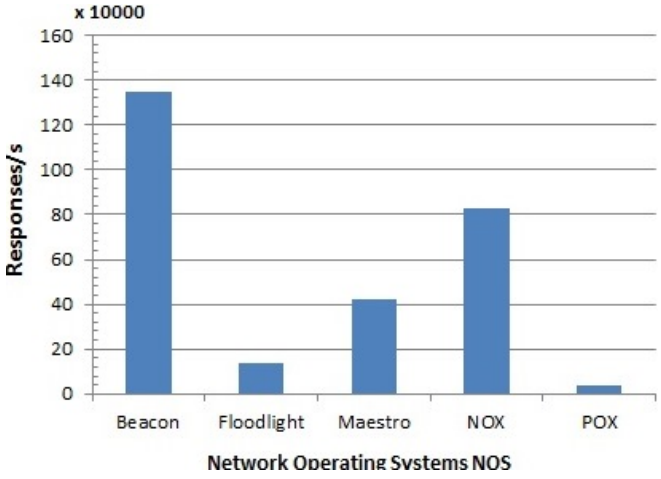


Fig. 2. Test of throughput of NOS [25]

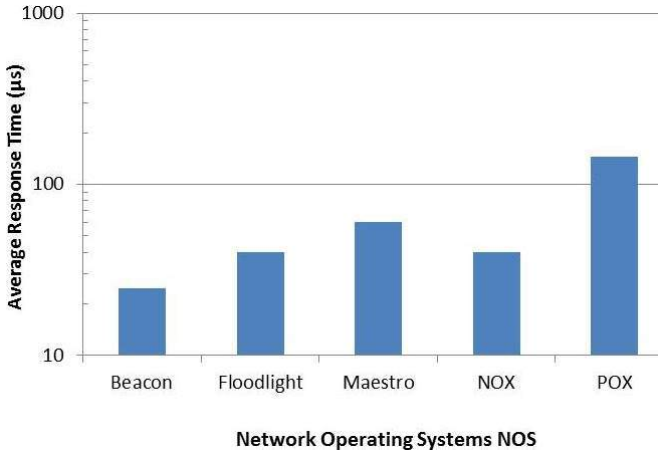


Fig. 3. Test of latency of NOS [25]

B. Resilience and Recovery

One of the problems related with the centralized control proposed by Openflow [2] is the vulnerability of the network. A failure of the controller can negatively compromise resilience of the whole network. Additionally, Openflow [2] establishes the configuration of one or more backup controllers, but doesnt provide a coordination mechanism between them.

CPRecovery component [12] is primary-backup mechanism that permits the replication between primary and secondary SDN controller in two phases: the *replication phase* and the *recovery phase*. The *replication phase* acts during normal functioning of the controller and send regular updates to the backup controller. The *recovery phase* acts during a failure state of the primary controller and starts the backup controller as a main controller. The failure state can be fired by abruptly aborting of a NOS process, an unexpected error of the application (API) or a DDoS attack. The results of experiments show a recovery time (from failure state to backup connection) of 800 ms.

C. Security

The current Internet architecture has huge problems to support, verify and enforce security properties. Typical Security Systems are based on securing hosts (e.g. antivirus) or installing specialized network devices (middleboxes) to detect anomalies into the network. However, these solutions become ineffective when the host is compromised (zero day vulnerabilities) or require constant updates and effort from the network administrator for example selecting the traffic to be filtered. In this point SDN can help to improve and develop new security models.

Pedigree [13] is a security system based on SDN for enterprise networks. This model uses Openflow [2] and a simple kernel OS-function on hosts to help the network controller to determine the provenance of the traffic network. Once the connection is validated, the controller enables and establishes the network path in switches. This system is designed with two components: the *tagger* and the *arbiter*. The *tagger* monitors all events in the OS and creates a tag for each process. When a process requires data sent through the network, a tagstream is sent to the controller. The controller has an *arbiter* function that checks the tags, verifies and orders an installment of the appropriate flowtables in switches. If the arbiter detects an unauthorized connection attempt, the controller orders a blockage of the flow of this link. Although, these processes generate additional load on the host and the network, the overhead is comparable to running an antivirus software.

D. Convergence and Management

SDN-Opeflow [2] architecture was originally developed for enterprise-campuses networks. However, to try and extend this architecture to large networks requires attend some issues, for example the problem of Interdomain Routing.

Today, the largest deployed protocol to communicate autonomous systems AS is the Border Gateway Protocol (BGP). BGP is a destination-based forwarding paradigm. In other words, the routing information between Autonomous Systems AS depends on the destination address in the IP packet header. An AS interchanges information and influence the traffic flow only with his direct neighboring AS. This makes it difficult to control of traffic flows along an entire path or in a remote part of the network. The task is to integrate SDN architecture to interconnect different AS.

In [15], an interdomain-routing solution based on BGP uses a NOX-Openflow [2] architecture. The new Inter-AS system proposes to extend two NOX components: *messenger* and the *switch*. *Messenger* creates a dedicated channel (port 2603) and enables the exchange of information between several NOX components. In other words, this *messenger channel* is used as a dependency of the Inter-AS component. The *Switch* adds a call to the Inter-AS component in case of an outside destination and selects the right datapath and port to forward the packet.

Another challenge to considerate is the integration with legacy networks. That is, networks with non-Openflow capacity (actual Internet infrastructure). It is noted that equipment could be upgraded or even replaced to support Openflow [2]. However, this change means costly network re-engineering. For this reason, the coordination between new SDN equipment and legacy infrastructure is necessary.

The project LegacyFlow [36] proposes a solution for coordination between Openflow [2] and traditional networks. The model introduces a new component between these architectures called *Legacy Datapath(LD)* that provides a bridge between different features of the legacy equipment. LD receives and interprets the messages sent by the Openflow controller and applies the corresponding actions in a real switch. Additionally, LegacyFlow [36] also proposes to add new actions to Openflow protocol [3] in order to improve the integration with traditional networks.

V. DISCUSSION AND CONCLUSIONS

This paper presents the state-of-the-art, the evolution and the challenges of Sofware Defined Networking in the last few years. Below, there is an exposition of some final comments and conclusions.

The separation of *control* and *data planes* opens the possibility to easily develop new services and network applications to industry and research community. Openflow [2] is a SDN technology based on the logic of match/action in *data plane* taking advantage of the actual network hardware capabilities. However, SDN can be expanded beyond match/action paradigm. For example, SDN can take into account several extensions, such as middleboxes or programmable custom packet processors.

This evolution could offer new services, for example on the fly encryption, transcoding, or traffic classification. Nevertheless, this requires a unifying control framework to allow coordination between the different types of network devices, as well as consensus and vendor support. At this point, the ONF [4] will take a decisive factor with the publication of new SDN protocols.

In *control plane*, the programming, debugging and testing are open issues in SDN. The High Level Languages facilitate the development of applications, but the composing and coupling of heterogeneous component are still difficult. For example, compose applications using NOX/POX [23] and Floodlight [26] or Beacon [25] simultaneously in the same controller is complicated today. The northbound policy languages also need to be constantly improved and tested to their implementation in production networks.

Finally, it is necessary to remark that Software Defined Networking is a tool. The research community can use this tool and develop new protocols, services, network applications taking advantage of the global view and the huge amount of information about the network.

ACKNOWLEDGMENT

This work was supported by the Agencia Española de Cooperación Internacional para el Desarrollo (AECID, Spain) through Acción Integrada MAEC-AECID MEDITERRÁNEO A1/037528/11.

Ángel Leonardo Valdivieso Caraguay and Lorena Isabel Barona López are supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program N. 2543-2012. The authors would also like to thank Ana Lucila Sandoval Orozco for her valuable comments and suggestions to improve the quality of the paper.

REFERENCES

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, (New York, NY, USA), pp. 1–12, ACM, August 2007.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, March 2008.

[3] "OpenFlow Switch Specification v.1.1.0," pp. 1–56, 2011.

[4] "Open Networking Foundation." https://www.opennetworking.org/.

[5] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A Network Virtualization Layer," tech. rep., OpenFlow Switch Consortium, October 2009.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the Production Network be the Testbed?," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, (Berkeley, CA, USA), pp. 1–6, USENIX Association, October 2010.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *Proceedings of the The 16th ACM SIGPLAN International Conference on Functional Programming*, (New York, NY, USA), pp. 279–291, ACM, September 2011.

[8] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High-Level Reactive Network Control," in *Proceedings of the First Workshop on Hot topics in Software Defined Networks*, (New York, NY, USA), pp. 43–48, ACM, August 2012.

[9] S. M. Patrick Le Callet and A. Perkis, "Qualinet White Paper on Definitions of Quality of Experience," *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)*, March 2012.

[10] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking," in *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks*, vol. 1, pp. 1–5, IEEE, September 2012.

[11] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proceedings of the 23rd International Teletraffic Congress*, pp. 1–7, ITCP, September 2011.

[12] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A Replication Component for Resilient OpenFlow-based Networking," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 933–939, IEEE, April 2012.

[13] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, "Securing Enterprise Networks Using Traffic Tainting," tech. rep., August 2009.

[14] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and Circuit Network Convergence with OpenFlow," in *2010 Conference on Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference (OFC/NFOEC)*, pp. 1–3, IEEE, March 2010.

[15] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An Inter-AS Routing Component for Software-Defined Networks," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 138–145, IEEE, April 2012.

[16] F. de Oliveira Silva, J. de Souza Pereira, P. Rosa, and S. Kofuji, "Enabling Future Internet Architecture Research and Experimentation by Using Software Defined Networking," *European Workshop on Software Defined Networking*, vol. 0, pp. 73–78, October 2012.
[17] B. Pfaff, J. Pettit, K. Amidon, and M. Casado, "Extending Networking into the Virtualization Layer," in *Proceedings of ACM SIGCOMM HotNets*, ACM, October 2009.
[18] P. S. Pisa, N. C. Fernandes, H. E. Carvalho, M. D. Moreira, M. E. M. Campista, L. H. M. Costa, and O. C. M. Duarte, "OpenFlow and Xen-Based Virtual Network Migration," in *Communications: Wireless in Developing Countries and Networks of the Future*, vol. 327, pp. 170–181, Springer Berlin Heidelberg, September 2010.
[19] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for. Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, (New York, NY, USA), pp. 19:1–19:6, ACM, October 2010.
[20] C. Elliott, "GENI: Opening Up New Classes of Experiments in Global Networking," *IEEE internet computing*, vol. 14, pp. 5–10, January 2010.
[21] "OFELIA." http://www.fp7-ofelia.eu/.
[22] C. Guimarães, "Extending and Deploying Ofelia in BRAzil (EDO-BRA)," February 2013.
[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.
[24] T. S. E. N. Zheng Cai, Alan L. Cox, "Maestro: A system for scalable openflow control," tech. rep., December 2010.
[25] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, (New York, NY, USA), pp. 13–18, ACM, August 2013.
[26] "Floodlight." http://www.projectfloodlight.org/.
[27] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.
[28] Z. Wan and P. Hudak, "Functional Reactive Programming from First Principles," in *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, (New York, NY, USA), pp. 242–252, ACM, May 2000.
[29] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with caps: Managing Usage Caps in Home Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, (New York, NY, USA), pp. 470–471, ACM, August 2011.
[30] "Frenetic." https://github.com/frenetic-lang/frenetic.
[31] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, (Berkeley, CA, USA), pp. 1–14, USENIX Association, April 2013.
[32] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A Demonstration of Fast Failure Recovery in Software Defined Networking," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44, pp. 411–414, Springer Berlin Heidelberg, June 2012.
[33] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, (New York, NY, USA), pp. 7–12, ACM, August 2012.
[34] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, (Berkeley, CA, USA), pp. 10–10, USENIX Association, April 2012.
[35] "Cbench." http://archive.openflow.org/wk/index.php/Oflops.
[36] F. Farias, J. Salvatti, E. Cerqueira, and A. Abelem, "A Proposal Management of the Legacy Network Environment Using Openflow Control Plane," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 1143–1150, IEEE, April 2012.

Ángel Leonardo Valdivieso Caraguay was born in Loja, Ecuador, in 1985. He received a B.S. degree in Electronics and Telecommunications Engineering from the Escuela Politécnica Nacional, Quito, Ecuador in 2009 and a M.S. degree in Information Technology from the University of Applied Sciences Hochschule Mannheim, Germany in 2012. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His research interests include computer networks and software-defined networking.

Lorena Isabel Barona López was born in Ambato, Ecuador, in 1985. She received a B.S. degree in Electronic and Information Networks Engineering from the Escuela Politécnica Nacional, Quito, Ecuador in 2010 and a M.S. degree in Telecommunications Engineering from the Universidad Politécnica de Madrid, Spain in 2013. She is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. Her research interests include computer communication and software-defined networking.

Luis Javier García Villalba (SM04) was born in Madrid, Spain, in 1969. He received the Telecommunication Engineering degree from the Universidad de Málaga, Spain, in 1993, the M.Sc. degree in computer networks in 1996, and the Ph.D. degree in computer science in 1999, the last two from the Universidad Politécnica de Madrid, Spain. He was a Visiting Scholar at the Research Group Computer Security and Industrial Cryptography (COSIC), Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium, in 2000, and a Visiting Scientist at the IBM Research Division (IBM Almaden Research Center, San Jose, CA) in 2001 and in 2002. He is currently an Associate Professor in the Department of Software Engineering and Artificial Intelligence at the Universidad Complutense de Madrid (UCM) and Head of the Complutense Research Group GASS (Group of Analysis, Security and Systems), which is located at the School of Computer Science at the UCM Campus. His professional experience includes research projects with Hitachi, IBM, Nokia, Safelayer Secure Communications, and VISA. His main research interests are in information security, computer networks, and software-defined networking. Dr. Garca Villalba is an Associate Editor in Computing for IEEE Latin America Transactions since 2004.

Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 735142, 10 pages
http://dx.doi.org/10.1155/2014/735142

Review Article

SDN: Evolution and Opportunities in the Development IoT Applications

Ángel Leonardo Valdivieso Caraguay, Alberto Benito Peral, Lorena Isabel Barona López, and Luis Javier García Villalba

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases s/n, Ciudad Universitaria, 28040 Madrid, Spain

Correspondence should be addressed to Luis Javier García Villalba; javiergv@fdi.ucm.es

Received 9 December 2013; Accepted 27 December 2013; Published 4 May 2014

Academic Editor: Young-Sik Jeong

The exponential growth of devices connected to the network has resulted in the development of new IoT applications and on-line services. However, these advances are limited by the rigidity of the current network infrastructure, in which the administrator has to implement high-level network policies adapting and configuring protocols manually and usually through a command line interface (CLI). At this point, Software-Defined Networking (SDN) appears as a viable alternative network architecture that allows for programming the network and opening the possibility of creating new services and more efficient applications to cover the actual requirements. In this paper, we describe this new technology and analyze its opportunities in the development of IoT applications. Similarly, we present the first applications and projects based on this technology. Finally, we discuss the issues and challenges in its implementation.

1. Introduction

The emergence of new services and applications on-line, both in fixed terminals and mobile devices, has made the communication networks a strategic point in companies, institutions, and homes. The continued evolution of these services and the growth of the information circulating the Internet, bring unanticipated challenges to developers and companies. The advances in micro-electro-mechanical systems (MEMS) increase the development of devices that automatically record, process and send information through the network. This kind of device, mainly consisting of sensors and actuators (RFID, Bluetooth Devices, Wireless Sensor Networks (WSN), Embedded Systems, and Near Field Communication (NFC)), has led to the origin of new ideas, concepts, and paradigms such as the Internet of Things (IoT).

This device uses different ways to connect to the network including the traditional network infrastructure. At present, there are 9 billion connected devices and a number of 24 billon is expected for 2020 [1]. This device uses different ways to connect to the network including the traditional network infrastructure. However, the traditional equipment and network protocols are not designed to support the high level of scalability, high amount of traffic and mobility. The current architectures are inefficient and have significant limitations to satisfy these new requirements.

The infrastructure responsible for transmitting the information coming of IoT devices (routers, switches, 3G and 4G networks, and access points) should be adapted to the new post-PC services (VoIP, sensor virtualization, QoS, cloud computing, and IoT applications) while providing security, stability, high rate, and availability amongst others. Some efforts such as the European Union Projects of SENSEI [2], Internet of Things-Architecture (IoT-A) [3], or Cognitive Management Framework for IoT [4] as well as the new protocols for Wireless Sensor Networks including the Data Driving Routing Protocol [5] have tried to get smarter connectivity between network elements (fully integrated Future Internet). However, these may not be the best options for a particular device or application domains (Smart Grid,

Intelligent Transportation, Smart Home, Health Care, and Environmental Monitoring and others). For this reason, in the last years, the idea of customizing the network behavior has emerged and gives users the flexibility to use the network resources according to their needs. Additionally, the development of new technologies to take decisions on IoT networks uses different calculation algorithms (genetic algorithms, neural networks, evolutionary algorithms, and other artificial intelligence techniques). It is desirable that these algorithms can be easily and dynamically implemented in the network equipment without waiting to be published in a protocol.

Software Defined Networking (SDN) is a network architecture that eliminates the rigidity present in traditional networks. Its structure allows the behavior of the network to be more flexible and adaptable to the needs of each organization, campus, or group of users. Besides, its centralized design allows important information to be collected from the network and used to improve and adapt their policies dynamically. The development in recent years has impulsed new concepts, such as the network operating system (NOS). NOS tries to emulate the progress in computer systems. In this paper, the evolution of the main NOS is also analyzed. With this tool, it is possible to test the SDN concept in multiple projects (Home Networking, Data centers, Security, Virtualization, and Multimedia among others). Similarly, SDN has led to the design of models that integrate and finally achieve convergence of commonly separate architectures (Wifi-4G-LTE). However, these opportunities are still far from being implemented globally in production. Important issues such as convergence with existing networks, scalability, performance, and security are the challenges that should be overcome to be positioned in the market.

In this piece of work, we describe SDN and its evolution in the last years and analyze the opportunities and challenges in the future for this technology. Specially, the development of new generations IoT application (Smart Environment). The work is structured as follows: in Section 2 the limitations of traditional networks is presented; next, Section 3 defines the Software-Defined Networking concept; Section 4 presents the evolution of Network Operating Systems (NOS); then, in Section 5, the first SDN applications are reviewed; in Section 6, the challenges of SDN technology are discussed; finally, Section 7 presents the conclusions.

2. Limitations of Traditional Architectures

The idea of transmitting information between two points through a network led to the design of communication protocols (TCP/IP, HTTPS, and DNS) and the creation of specialized devices in the transmission of information. These devices have evolved resulting in a variety of equipment (hub, switch, router, firewall, IDS, middlebox, and filters). This development has produced an exponential increase in the number of connected devices, in addition to the increase of the transmission rate and the emergence of online services (e-banking, e-commerce, e-mail, VoIP, etc.).

All the devices responsible for transmitting information have similar features in their design and manufacture. First, there is a specialized hardware in the packet processing (data plane), and over the hardware works an operating system (usually Linux) that receives information from the hardware and runs a software application (control plane). The software contains thousands of lines of code for determining the next hop that a packet should be taken in order to reach its destination. The program follows the rules defined by a specific protocol (there are currently about 7000 RFCs) or some proprietary technology of the vendor. Modern equipment also analyzes information packets to search malicious information or intrusions (firewalls and IDS). However, all technology or software used in the manufacturing of these devices is rigid or closed to the network administrator. The administrator is limited only to configure some parameters, usually through low level commands using a command line interface (CLI). Moreover, each node is an autonomous system which finds the next hop to be taken by a packet to reach its destination. Some protocols (OSPF, BGP) allow the nodes to share control information between them, but only with its immediate neighbors and in a limited way in order to avoid tradditional load on network. This means that there is not a global view of the network as a whole. If the users need to control and modify a particular path, the administrator has to test with parameters, priorities, or uses gadgets to achieve the expected behavior in the network. Each change in the network policy requires individual configuration directly or remotely from each of the devices. This rigidity makes the implementation of high-level network policies difficult. Moreover, the policies require to be adaptive and dynamically react according to the network conditions.

As Operating Systems (OS) evolve and adapt to the new needs and technological trends (support multi-CPU, multi-GPU, 3D, touch screen support, etc.), the network adaptability to new requirements (VLAN, IPv6, QoS, and VoIP) is implemented through protocols or RFCs. However, in the operating system the separation between hardware and software allows the continuous update of application, or even the reinstallation of a new version of an OS. In the area of networks, the design and implementation period of a new idea could take several years until it is published in a protocol and incorporated in new devices. Some services are proprietary of the vendors and require that all network infrastructure belong to the same vendor to work properly. This limitation brings on the dependence on a specific technology or vendor.

3. Software-Defined Networking SDN

The concept of Software-Defined Networking is not new and completely revolutionary; rather it arises as the result of contributions, ideas, and developments in research networking. In [6], three important states are determined in the evolution of SDN: Active Networks (mid-90s to early 2000), separation of data and control planes (2001–2007), and the OpenFlow API and NOS (2007–2010). All these aspects are discussed below.

The difficulty for researchers to test new ideas in a real infrastructure and the time, effort and resources needed to standardize these ideas on the Internet Engineering

Task Force (IETF) necessarily give some programmability to network devices. Active networks offer a programmable network interface or API that opens the individual resources of each node for the users, such as processing, memory resources, and packet processing and includes personalized features for the packets that circulate through the node. The need to use different programming models in the nodes was the first step for research in network virtualization, as well as the development of frameworks or platforms for the development of application on the node. The Architectural Framework for Active Networks v1.0 [6, 7] contains a shared Node Operating System (NodeOS), a set of execution environments (Execution Environments (EEs)), and active applications (Active Applications (AAs)). The NodeOS manages the shared resources, while the EE defines a virtual machine for the packet operations. The AA operates within an EE and provides the end-to-end service. The separation of packets to each EE depends on a pattern in the header of incoming packets to the node. This model was used in the PlanetLab [8] platform, where researchers conducted experiments in virtual execution environments and packets were demultiplexed to each virtual environment based on its header. These developments were important, especially in the investigation of architectures, platforms, and programming models in networks. However, their applicability in industry was limited and mainly criticized for its limitations in performance and safety. The work presented in [9] is an effort to provide the best performance to the active networks, and the Secure Active Network Environment Architecture [10] tried to improve their security.

The exponential growth in the volume of traffic over the network produces the necessity to improve the supervision process and uses best management functions such as the management of paths or links circulating the network (traffic engineering), prediction traffic, reaction, and fast recovery if there are network problems, among others. However, the development of these technologies has been strongly limited by the close connection between the hardware and software of networking devices. Besides, the continuous increase in link rates (backbones) means that the whole transmission mechanism of packets (packet forwarding) is focused on the hardware, separating control, or network management to an application of software. These applications work best on a server, because it has higher processing and memory resources compared with a single network device. In this sense, the project ForCES (Forwarding and Control Element Separation) [11] standardized by the IETF (RFC 3746) established an interface between data and control plane in the network nodes. The SoftRouter [12] used this software interface to install forwarding tables in the data plane of routers. Additionally, the Routing Control Platform (RCP) [13] project proposed logical centralized control of the network, thus facilitating the management, the innovation capacity, and programming of network. RCP had an immediate applicability because it uses an existing control protocol BGP (Border Gateway Protocol) to install entries in the routing tables of the routers.

The separation of data plane and the control plane allows the development of "clean-slate" architectures, such as the 4D project [14] and Ethanet [15]. 4D architecture proposes architecture of four layers based on functionality: data plane, discovery plane, dissemination plane, and decision plane. Moreover, the Ethanet project [15] proposes a centralized control system of links to business networks. However, the need for custom switches based on Linux, OpenWrt, NetFPGA with support for Ethane protocol made it difficult the applicability of this project. At the present time, the OpenFlow protocol [16] is the most widely used in the research community and it has been the basis of different projects. Companies like Cisco have also submitted a proposal for a new architecture called Cisco Open Network Environment (Cisco ONE).

Simplifying the previous analysis, the term Software-Defined Networking proposes some changes to the networks of today. First, the separation or decoupling of the data plane and control plane, allowing evolution and development independently. Secondly, it proposes a centralized control plane, thus having a global view of the network. Finally, SDN establishes open interfaces between the control plane and data plane. The differences between these architectures are shown in Figure 1.

The programmability of the network provided by SDN can be compared with the mobile applications running on an Operating System (Android and Windows Mobile). These applications use the resources of the mobile (GPS, accelerometer, and memory) through the API provided by the OS. Likewise, the network administrator can manage and program resources in the network, according to user needs, through available APIs (proprietary or open) on the controller.

3.1. OpenFlow. OpenFlow [16] was originally proposed as an alternative for the development of experimental protocols on university campus, where it is possible to test new algorithms without disrupt or interfere with the normal operation of traffic of other users. Nowadays, the Open Networking Foundation (ONF) [6] is the organization responsible for the publication of the OpenFlow protocol and other protocols for SDN, such as OF-Config [17].

The advantage of OpenFlow, compared with previous SDN protocols, is the use of elements and features of hardware available in most network devices. These elements are the routing tables and the common functions are as follows: read the header, send the packet to a port, and drop a packet, among others. OpenFlow opens up these elements and functions; so these can be controlled externally. This implies that, with a firmware update, the actual hardware could potentially support OpenFlow. The companies do not need a complete change of their hardware to implement SDN in their products and services.

The OpenFlow architecture proposes the existence of a controller, a switch OpenFlow, and a secure protocol of communication. These elements are shown in Figure 2. Each OpenFlow switch consists of flow tables that are managed by the controller. Each flow table has three elements: packet header, actions, and statistics. The packet header is like a mask that select the packets which will be processed by the switch. The fields used for comparison can be from

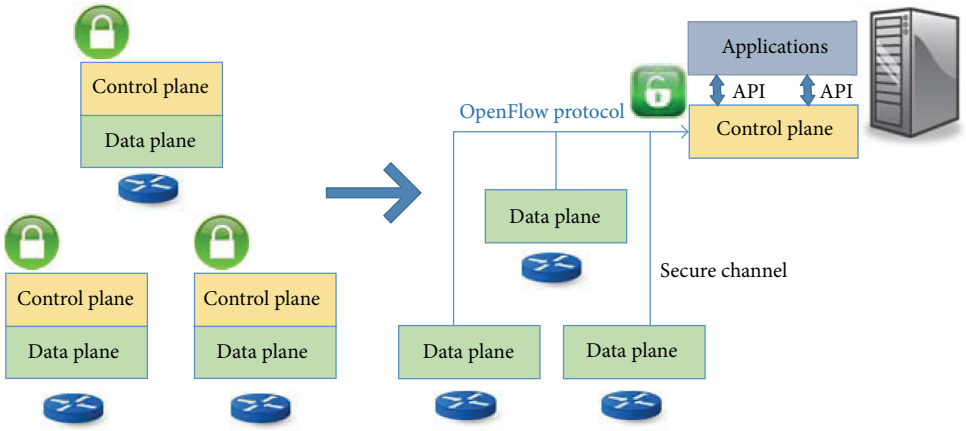


Figure 1: Comparison between traditional and SDN architectures.

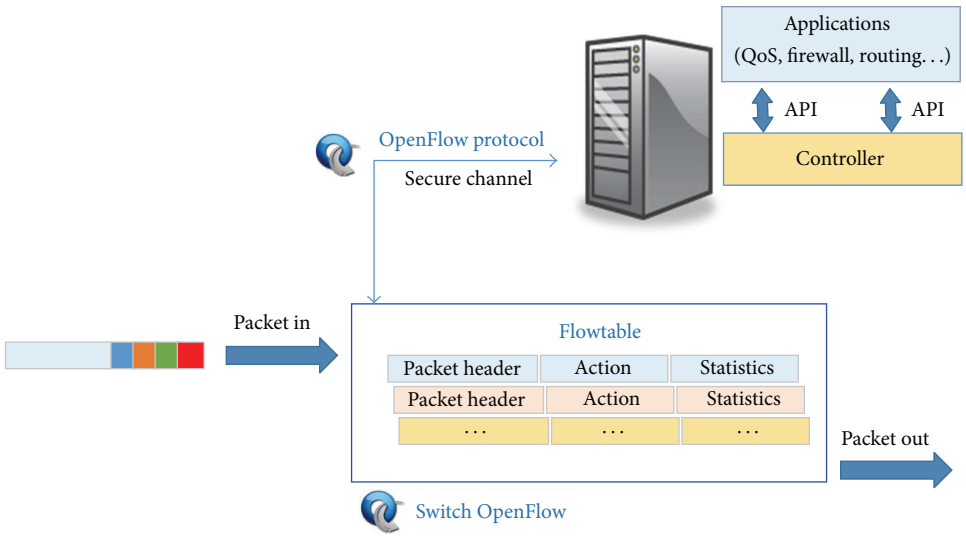


Figure 2: Elements of the OpenFlow architecture.

layer 2, 3, or 4 of the TCP/IP architecture. That means that there is not a separation between layers as in current architectures. All packets processed by the switch are filtered through this method. The number of fields that the switch can process depends on the version of the OpenFlow protocol. In OpenFlow v1.0 [18] (the most used version), there are 12 fields, while the latest version OpenFlow v1.3 defines the existence of 40 fields including support for IPv6.

Once the header of an incoming packet matches the packet header of the flow table, the corresponding actions for that mask are performed by the switch. There are main and optional actions. The main actions are as follows: forward the packet to a particular port, encapsulate the packet and send it to the controller, and drop the packet. Some optional actions are as follows: forward a packet through a queue attached to a port (enqueue action) or 802.1D processing capabilities. If the header of an incoming packet does not match with the packet header of the flow table, the switch (according to its configuration) sends the packet to the controller for its analysis and treatment. Finally, the statistics field uses counters to collect statistic information for administration purposes.

The OpenFlow protocol defines the following types of messages between the switch and the controller: controller to switch, symmetric, and asynchronous. The messages type controller to switch manage the state of the switch. Symmetric messages are sent by the controller or switch to initiate the connection or interchange of messages. The asynchronous messages update the control of the network events and the changes of state switch. Similarly, OpenFlow establishes two types of switches: OpenFlow-only and OpenFlow-enabled. OpenFlow-only switches use only OpenFlow protocol to process packets. On the other side, OpenFlow-enabled switches can additionally process the packet using traditional algorithms of switching or routing.

The controller receives the information from the various switches and remotely configures the flow tables of the switch. Here, the user can literally program the behavior of

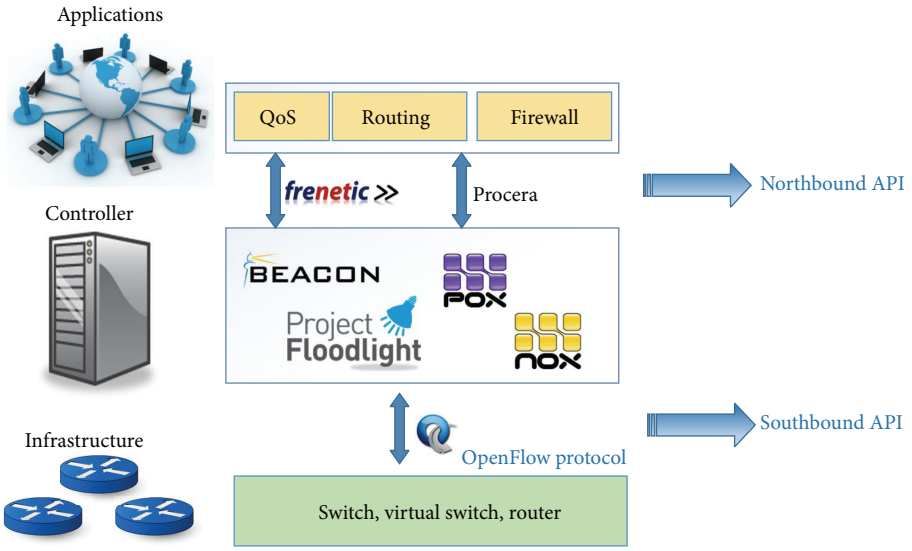


Figure 3: NOS, southbound, and northbound interfaces.

the network. Unlike active networks, which proposed a "Node Operating System." OpenFlow opens the notion of a Network Operating System (NOS). In this respect, in [19], the NOS is defined as the software that abstracts the installation of the state in the switches of network of the logic and applications that control the behavior of the network. In recent years, the NOS has evolved according to the needs and applications for researchers and network administrators.

4. Network Operating Systems Evolution (NOS)

The concept of Network Operating Systems (NOS) is based on the function of an operating system in computing. That is, the Operating System allows user to create applications using high-level abstraction of information, resources, and hardware. In SDN, some authors [20–22] have classified the abstractions of network resources as southbound and northbound interfaces (Figure 3). The function of the southbound interfaces is to abstract the functionality of the programmable switch and connect it to the controller software. A clear example of southbound interface is OpenFlow. On the southbound interfaces, you run a Network Operating Systems. An example of NOS is NOX [23–25], among others. On the other hand, the northbound interfaces allow applications or high-level network policies to be easily created and they transmit these tasks to Network Operating System (NOS). Examples of these interfaces are Frenetic [26, 27], Procera [21, 28], Netcore [29], and McNettle [30]. Then, they are analyzed in the main NOS and northbound interfaces.

The NOX software [23] is the first NOS for OpenFlow and consists of 2 elements: processes of controller (controller) and a global view of the network (network view). Depending on the current state of the network, the user can make decisions and set the network behavior through these processes. In NOX, traffic is handled at the level of flows (flow-based granularity); that is, all packets with the same header are treated similarly. The controller inserts, deletes entries, and reads the counters found in the flow tables of the switches. Furthermore, due to the dynamic nature of traffic, NOX uses events (event handlers) that are registered with different priorities to be executed when a specific event occurs in the network. The most used events are switch join, switch leave, packet received, and switch statistics received. Additionally, NOX includes "system libraries" implementations and common network services. Finally, NOX is implemented in C++ providing high performance. Moreover, there is an implementation entirely in Python denominated POX, which provides a more friendly developed language.

Beacon [24] is a Java-based OpenFlow controller. Its interface is simple and unrestricted; that is, the user can freely use the constructors available in Java (threads, timers, sockets, etc.). Furthermore, Beacon is a NOS based on events; that is, the user sets the events that the controller listens to. The interaction with OpenFlow messages of the switch is done by the library OpenFlowJ, an implementation of the OpenFlow 1.0 [18] protocol and IBeaconProvider interface that contains the following listeners: IOFSwitchListener, IOFInitializerListener, and IOFMessageListener. Additionally, Beacon has multithreading support and provides important APIs implementations (Device Manager, Topology, Routing, and Web UI) as well as the ability to start, add, and complete applications without completely terminating a process in Beacon (runtime modularity).

Although a NOS can handle the flow tables of the switches, there are some problems that can cause malfunction of the network [20, 22, 31]. For example, the controller receives the first packet that arrives at the switch and has not matched a header in the flow table. Then the controller analyzes it, assigns actions, and forwards these instructions to the switch so that the other similar packages follow the same route. However, during this time, the second, third, or fourth similar packets can be received by the controller and cause an erratic operation. In other words, there are

virtually two processes running, one on the controller and another on the switch, and these processes are not fully synchronized.

Another limitation is the composition; that is, if the user wants to configure two different services on the same switch (e.g., routing and monitoring), it is necessary to manually combine the two actions on the switch, prioritize, and keep the semantics of each element of the network. This makes the design, coordination, and reuse of the libraries very difficult. Additionally, the switch has to handle two types of messages simultaneously: packets and control messages. Any mismatch can cause a packet to be processed with an invalid policy and thereby causing major security problem on the network. For example, if there are two entries in a flow table with the same priority, the switch behavior might be nondeterministic, because the execution would depend on the design of the switch hardware. For this reason, the research community has worked on secure interfaces that automatically interact and coordinate the correct behavior of the switch (northbound).

Procera [21, 28] is a framework that allows politics or high-level network configurations to be expressed. This architecture provides different actions and control domains to program the behavior of the network. The main domains of control are as follows: time, data usage, flow, and status. With these domains, the user can determine a behavior depending, for example, on the time of day, amount of data transmitted, privileges or groups of users, type of transmitted traffic, and so forth. Actions can be temporal or reactive and are expressed on a high-level language based on Functional Reactive Programming (FRP) and Haskell. In [21] are the details of this language as well as examples of using Procera in monitoring applications and users control on a college campus.

Frenetic [26, 27] is a high-level language dedicated to SDN networks developed in Python. It is structured by 2 sublanguages: a Network Query Language and a Reactive Network Policy Management Library. The Network Query Language allows the user to read the status of the network. This task is performed by installing rules (low-levels rules) on the switch which does not affect the normal operation of the network. In addition, the Network Policy Management Library is designed based on a language for robots, Yampa, [32] and web programming libraries in Flapjax [33]. The actions use a constructor type rule containing a pattern or filters and action list as arguments. The main actions are as follows: sending to a particular port, sending packet to the controller, modification the packet header, and blank action that is interpreted as discard the packet. The installation of these policies is performed by generating policy events (queries), primitive events (Seconds, SwitchJoin SwitchExit, and PortChange), and listener (Print and Register). The results of experiments [26] show that Frenetic provides simplicity and a significant savings in code and lower consumption of network resources compared to NOX.

One of the additional advantages of this language is the composition; that is, independent functional modules can be written and the runtime system coordinates its proper function in the controller and the switch. There are 2 types of composition: sequential and parallel. In sequential composition, the output of one module is the input of the next, for example, a load balancer that first modifies the IP destination of a packet and then searches the output port according to the new IP header. In parallel composition, both modules are executed virtually simultaneously in the controller; for example, if the balancer sends a packet with destination IP A to port 1, and packet B IP destined to port 2, this composition would result in a function that sends incoming packets for ports 1 and 2.

McNettle [30] is a controller specially designed to offer high scalability at the SDN network. This is achieved using a set of message handlers (one for each switch) having a function that handles the switch-local and network-state variables and manages the supply actions from the network flows. The idea is that the messages from the same switch are handled sequentially, while messages from different switches are handled concurrently. Similarly, each message is processed in a single core CPU to minimize the number of connections and synchronizations inter-cores among other performance improvements. The tests performed in [30] show that McNettle have a higher multicore performance compared to NOX or Beacon.

The controller proposed in [31] is based on the verification of the established politics, instead of searching bugs monitoring the controller operation. To perform the verification, the first step is to make use of the high-level language Netcore [29] to describe only the network behavior. Then, the Netcore Compiler translates the politics to network configurations as flow table entries. The flow tables information is analyzed by the Verifier Run-time System which transforms the network configuration into a lower abstraction level named Featherweght OpenFlow. Featherweght Openflow is a model that use synchronization primitives to guarantee the coherent behavior of the flowtables. Additionally, the Kinetic tool is described in [22]; this tool allows performing consistent updates in the network using two mechanisms: per-packet consistency and per-flow consistency. The per-packet consistency mechanism ensures that a packet that is transmitted across the network is processed with the same configuration when an update occurs. The per-flow consistency mechanism ensures that every packet that belongs to the same flow (e.g., a TCP connection) will be processed in the same way by every switch in the network.

5. First SDN Applications

Software-Defined Networking provides the ability to modify the network behavior according to user needs. In other words, SDN itself doesnt solve any particular problem, but provides a more flexible tool to improve the network management. In order to test the advantages of this architecture, the research community has presented multiple projects of interest. Next, some of these applications are described.

5.1. Home Networking. In the emerging topic of Internet of Things (IoT), the management of devices and network

resources in home networks is a big challenge due to the number of users and devices connected to the same point (usually an access point). In [21, 34], the authors present an implementation of an OpenFlow-based system that allows the monitoring and management of user and control of the Internet access based on "usage caps" or a limited data capacity for each user or device. The system provides visibility of the network resources and management of access based on user, group, device, application, or time of day and even enables the ability to exchange data capacity with another user. The system of control and network monitoring uses the friendly interface Kermit. The capacity management and network policies are based on the Resonance language [35].

5.2. Security. The global vision of the network can improve the security of the systems. This security cannot be based only in the host-security, because such defenses are ineffective when the host is compromised. In [36], the Pedigree system is presented as an alternative to provide security in the traffic moving in an enterprise network. This OpenFlow-based system allows to the controller the analysis and the approval of connections and traffic flows in the network. The host has a security module in the kernel (tagger) that is not under users control. This module labels the connections request to send information through the network (processes, files, etc.). This label is sent to the controller (arbiter) in the start of the communication. The controller analyzes the tagger and accepts or rejects the connection according to its policies. Once the connection is authorized, the corresponding flow tables are installed in the switch. Pedigree increases the tolerance to a variety of attacks, such as polymorphic worms. The systems increase the load in the network traffic and the host. However, this load is not higher than common antivirus software.

5.3. Virtualization. The concept of virtualization in networks is similar to OS-virtualization, where different Operating Systems can share hardware resources. That is, in network virtualization, it is intended that multiple virtual networks can operate on the same infrastructure, each with its own topology and routing logic. Initially, VLAN technologies and private virtual networks allow the different users to share network resources. However, the separation is controlled only by the network administration and with limited parameters (port number) and just work with known network protocols. With the SDN data-control separation, the possibilities to create new advanced virtual networks are promising. For example, Flowvisor [37, 38] is an OpenFlow-based project that allows creating slices based on multiple parameters, such as bandwidth, flowspace (src/dst MAC, src/dst IP, and src/dst TCP ports), or CPU switch load. Each slice is independent; that means that it does not affect the traffic of the others slices. Additionally, it is possible to subdivide slices in order to create hierarchical models. A network service that takes advantage of virtualization of network resources is the migration of virtual networks. In [39], a system that enables the migration of the switch configuration to another device into the network is proposed, but without disrupting the active network traffic. The controller copies the flow tables configuration from the old to the new switch and modifies the paths automatically. This service enables the possibility to replace a network device avoiding the disruption or packet loss. This advantage can be used to dynamically modify the resources used in the network (green networks). In other words, the network can turn off or disable the unnecessary devices (nights or weekends) and automatically enable them in function of the traffic demand (peak hours).

5.4. Mobile Networks. The devices in the infrastructure on mobile carrier networks share similar limitations as computer networks. Likewise, the carrier networks execute standards and protocols, for example, the Third Generation Partnership Project (3GPP) as well as the private vendors implementations. At this point, the SDN paradigm and its flow-based model can be applied on this kind of infrastructure offering better tools. Software-Defined Mobile Network (SDMN) [40] is an architecture that enables openness, innovation, and programmability to operators, without depending on exclusive vendors or over the top (OTT) service providers. This model consists of two elements: MobileFlow Forwarding Engine (MFFE) and the MobileFlow Controller (MFC). MFFE is a simple and stable data plane and with high performance. It has a more complex structure than an OpenFlow switch, because it must support additional carrier functions, such as layer 3 tunneling (i.e., GTP-U and GRE), access network nodes functions, and flexible charging. The MFC is the high performance control plane, where the mobile networks applications can be developed. Additionally, MFC has 3GPP interfaces to interconnect with different Mobile Management Entities (MMEs), Serving Gateways (SGWs), or Packet Data Network Gateways (PGWs).

5.5. Multimedia. The multiple online multimedia services, for example, the real time transmissions, require high levels of efficiency and availability of the network infrastructure. According to studies presented by CISCO, the IP video traffic will grow from 60% in 2012 to 73% by 2017 [41]. Moreover, in the last years, the concept of Quality of Experience (QoE) [42] gained particular strength, which attempts to redefine the Quality of Service (QoS) considering the level of user acceptance to a particular service or multimedia application. Therefore, SDN allows the optimization of the multimedia management tasks. For example, in [43] is improved the QoE experience through the path optimization. This architecture consists of two elements: the QoS Matching and Optimization Function (QMOF) that reads the different multimedia parameters and establishes the appropriate configuration for this path, and the Path Assignment Function (PAF) that regularly updates the network topology. In case of degradation of the quality on the links, the system automatically modifies the path parameters taking in count the priorities of the users. Similarly, the project OpenFlow-assisted QoE Fairness Framework (QFF) [44] analyzes the traffic in the network and identifies the multimedia transmissions in order to optimize

them in function of the terminal devices and the network requirements.

5.6. Reliability and Recovery. One of the most common problems in the traditional networks is the hardness to recover a link failure. The convergence time is affected by the limited information of the node to recalculate the route. In some cases, it is necessary the intervention of the network administrator to reestablish the network datapath. At this point, the global vision of SDN enables the customizing of recovery algorithms. [45] proposed an OpenFlow-based system that uses the mechanism of restoration and protection to calculate an alternative path. In restoration mechanism, the controller looks for an alternative path when the fail signal is received. Meanwhile, in protection the system anticipates a failure and previously calculates an alternative path. Similar to a failure on switch or routers, the malfunction of the SDN controller (NOS failure, DDoS attack, and application error) can cause a collapse of the whole network. Therefore, the reliability of the network can be ensured through backup controllers. However, it is necessary to coordinate and update the information of control and configuration between principal and backup controllers. The CPRecovery [46] component is a primary backup mechanism that enables the replication of information between primary and backup controller. The system uses the replication phase to maintain the updated backup controller and the phase of recovery that starts the controller backup at the moment it detects a failure of the principal controller.

6. Challenges of SDN Technology

The SDN advantages as applied technology in production networks are still close but not immediate. Furthermore, there are some challenges in terms of security, scalability, and reliability, among other aspects, which must be overcome in order to be considered acceptable for commercial users. Next, these aspects are analyzed.

As it was previously explained, the separation between data and control plane enables their independent development and evolution. In data plane, the rate of packet processing depends on the used hardware technology, such as *Application-Specific Integrated Circuits (ASIC), Application-Specific Standard Products (ASSP), Field Programmable Gate Array (FPGA), or multicore CPU/GPP*. Meanwhile, in control plane the performance also depends principally on the hardware and the NOS (Beacon, POX, Floodlight). However, a poor performance of one of the two levels can cause significant problems, such as packet loss or delay and incorrect behavior of the network of denial of service DDoS. For this reason, a balance in performance, cost, and facility of development for the hardware and software of SDN components is necessary.

Moreover, OpenFlow uses the common hardware resources of actual networks, such as the flow tables. However, SDN can be extended beyond flow tables and use additional resources offered by actual hardware [17]. The integration and research of new features between control and data plane is a recently open topic. Applications like encryption, analysis, and traffic classification and devices such as middleboxes and custom packet processors can be integrated and efficiently used by the SDN technology. On the other hand, the number and the position of the controllers in a network are open questions. The analysis presented in [47] exposes that the determining factors for the selection of the number and position are the topology and the expected performance of the network.

The security is another fundamental aspect that must also be taken into account. For example, not all network applications should have the same access privileges [20]. The assignment of profiles, authentication, and authorization to access the network resources are necessary. In addition, OpenFlow establishes the optional use of TLS (Transport Layer Security) as authentication tool between switch and controller. However, there are not clear specifications that provide security for multiple controller systems that interchange information among them and with the switches. Additionally, Openflow establishes that an unknown packet could be send completely (or its packet header) to the controller, it can easily be affected by DDoS attacks by the sending of multiple unknown packets to the switch.

The transition between actual network architectures to SDN-based architectures is also an open issue. Despite the emergence of network devices with OpenFlow support (IBM and NEC) in the market, it is impossible to replace the network infrastructure completely. The transition period requires mechanism, protocols, and interfaces allowing coexistence between both architectures. Currently, there are important efforts to achieve this objective; the Open Networking Foundation (ONF) published the IF-Config Protocol [48] as a first step to the configuration of OpenFlow devices. Similarly, the European Telecommunications Standards Institute (ETSI) as well as the el IETFs Forwarding and Control Element Separation Working Group (ForCES) works on the standardization of interfaces for the appropriated development of this technology.

7. Conclusion

The exponential growth of devices and online services that exchange information over the network consolidated the concept of Internet of Things (IoT). In this new approach, the rigidity of traditional architectures is inefficient suggesting rethinking new ways to use the infrastructure and technology communications. Software-Defined Networking has emerged as an alternative to the current problems of traditional networks. It allows administrators to have a global view of the network, as well as the opportunity to control the network according to the needs of each organization.

This work presents the basis of this technology and the development of Network Operating Systems NOS, as well as some interesting projects based on this paradigm. Additionally, the problems and challenges for the implementation of SDN in production networks are analyzed. It is noteworthy that SDN provides the tools to improve the management of the network behavior. The use of this tool and the development of new SDN applications are new fields of study. In the future, this paradigm will bring new ways

of viewing and using the communication networks as well as new business models focused on offering services and network applications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

Part of the computations of this work were performed in EOLO, the HPC of Climate Change of the International Campus of Excellence of Moncloa, funded by MECD and MICINN. This is a contribution to CEI Moncloa. Ángel Leonardo Valdivieso Caraguay and Lorena Isabel Barona López are supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program no. 2543-2012. The authors would also like to thank Ana Lucila Sandoval Orozco for her valuable comments and suggestions to improve the quality of the paper.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): a vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] SENSEI, Integrated EU Project, http://www.ict-sensei.org/.

[3] European Lighthouse Integrated Project, Internet of Things Architecture, http://www.iot-a.eu/public/.

[4] P. Vlacheas, R. Giaffreda, V. Stavroulaki et al., "Enabling smart cities through a cognitive management framework for the internet of things," in *IEEE Communications Magazine*, vol. 51, pp. 102–111, 2013.

[5] L. Shi, B. Zhang, H. T. Mouftah, and J. Ma, "DDRP: an efficient data-driven routing protocol for wireless sensor networks with mobile sinks," *International Journal of Communication Systems*, vol. 26, no. 10, pp. 1341–1355, 2013.

[6] Open Networking Foundation, https://www.opennetworking.org/.

[7] K. Calvert, *Architectural Framework for Active Networks Version 1.0*, 1999.

[8] PlanetLab, https://www.planet-lab.org/.

[9] T. Wolf and J. S. Turner, "Design issues for high-performance active routers," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 3, pp. 404–409, 2001.

[10] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A secure active network environment architecture: realization in SwitchWare," *IEEE Network*, vol. 12, no. 3, pp. 37–45, 1998.

[11] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) framework," RFC 3746, 2004.

[12] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The soft- router architecture," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[13] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI '05)*, vol. 2, pp. 15–28, USENIX Association, May 2005.

[14] A. Greenberg, G. Hjaimtysson, D. Maltz et al., "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[15] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker, "Ethane: taking control of the enterprise," in *Proceedings of the ACM SIGCOMM 2007: Conference on Computer Communications*, vol. 37, pp. 1–12, August 2007.

[16] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, 2008.

[17] A. Valdivieso, L. Barona, and L. Villalba, "Evolution and challenges of software defined networking," in *Proceedings of the 2013 Workshop on Software Defined Networks for Future Networks and Services*, pp. 61–67, IEEE, November 2013.

[18] OpenFlow Switch Specification v.1.0.0, December 2009.

[19] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: an intellectual history of programmable networks," Technical Report, Princeton University, Princeton, NJ, USA, 2013.

[20] S. Sezer, S. Scott-Hayward, P. K. Chouhan et al., "Are we ready for SDN? Implementation challenges for software-defined networks," in *IEEE Communications Magazine*, vol. 51, pp. 36–43, 2013.

[21] H. Kim and N. Feamster, "Improving network management with software defined networking," in *IEEE Communications Magazine*, vol. 51, pp. 114–119, 2013.

[22] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 323–334, 2012.

[23] N. Gude, T. Koponen, J. Pettit et al., "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[24] D. Erickson, "The beacon OpenFlow controller," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, pp. 13–18, ACM, August 2013.

[25] Floodlight, http://www.projectfloodlight.org/.

[26] N. Foster, R. Harrison, M. J. Freedman et al., "Frenetic: a network programming language," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[27] N. Foster, A. Guha, M. Reitblatt et al., "Languages for software defined networks," in *IEEE Communications Magazine*, vol. 51, pp. 128–134, 2013.

[28] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the 1st Workshop on Hot topics in Software Defined Networks*, pp. 43–48, ACM, August 2012.

[29] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 217–230, 2012.

[30] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 289–290, ACM, 2012.

[31] A. Guha, M. Reitblatt, and N. Foster, "Machine-verified network controllers," in *ACM SIGPLAN NOTICES*, vol. 48, pp. 483–494, ACM, 2013.

[32] A. Courtney, H. Nilsson, and J. Peterson, "The Yampa arcade," in *Proceedings of the ACM SIGPLAN 2003 Haskell Workshop*, pp. 7–18, ACM, August 2003.

[33] L. A. Meyerovich, A. Guha, J. Baskin et al., "Flapjax: a programming language for ajax applications," *ACM SIGPLAN Notices*, vol. 44, no. 10, pp. 1–20, 2009.

[34] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with caps: managing usage caps in home networks," in *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*, vol. 41, pp. 470–471, August 2011.

[35] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*, pp. 11–18, ACM, August 2009.

[36] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, *Securing Enterprise Networks Using Traffic Tainting*, 2009.

[37] R. Sherwood, G. Gibb, and M. Kobayashi, "Carving research slices out of your production networks with OpenFlow," in *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 129–130, ACM, 2010.

[38] R. Sherwood, G. Gibb, K. K. Yap et al., "Flowvisor: a network virtualization layer," in *OpenFlow Switch Consortium*, 2009.

[39] P. S. Pisa, N. C. Fernandes, H. E. Carvalho et al., "OpenFlow and Xen-based virtual network migration," in *Wireless in Developing Countries and Networks of the Future*, vol. 327, pp. 170–181, Springer, Berlin, Germany, 2010.

[40] K. Pentikousis, Y. Wang, and W. Hu, "MobileFlow: toward software-defined mobile networks," in *IEEE Communications Magazine*, vol. 51, pp. 44–53, 2013.

[41] The Zettabyte EraTrends and Analysis, May 2013.

[42] S. M. Patrick Le Callet and A. Perkis, "Qualinet white paper on definitions of quality of experience," in *Proceedings of the European Network on Quality of Experience in Multimedia Systems and Services, COST Action IC, 1003*, March 2012.

[43] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks*, vol. 1, pp. 1–5, IEEE, September 2012.

[44] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using OpenFlow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Humancentric Multimedia Networking (FhMN '13)*, pp. 15–20, ACM, August 2013.

[45] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A demonstration of fast failure recovery in software defined networking," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44, pp. 411–414, Springer, Berlin, Germany, 2012.

[46] P. Fonseca, R. Bennesby, E. Mota, and E. Passito, "A replication component for resilient OpenFlow-based networking," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 933–939, IEEE, April 2012.

[47] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *ACM SIGCOMM Computer Communication Review*, vol. 42, pp. 473–478, ACM, 2012.

[48] OpenFlow Management and Configuration Protocol (OF-Config) v.1.1.1, March 2013.

Computer Networks 92 (2015) 369–379

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Framework for optimized multimedia routing over software defined networks

Ángel Leonardo Valdivieso Caraguay*, Jesús Antonio Puente Fernández, Luis Javier García Villalba

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases 9, Ciudad Universitaria, Madrid 28040, Spain

ARTICLE INFO

Article history:
Received 27 October 2014
Revised 25 May 2015
Accepted 15 September 2015
Available online 25 September 2015

Keywords:
Multimedia
Quality of service
OpenFlow
SDN
Video streaming

ABSTRACT

The increase of data traffic moving in the Internet is particularly concentrated in multimedia content. However, the rigidity of the typical Internet architecture based on best-effort oriented IP network has limited the development of innovative network services. The new post-PC multimedia services require networks with high levels of flexibility and customization in order to provide efficient security, mobility, availability and QoS. In this context, software defined networking (SDN) is a novel paradigm that decouples the data and control plane in network devices and opens the programming capabilities of the network behavior. SDN opens the possibility to customize the network behavior in function of the different users requirements.

In this paper, we describe the SDN architecture and analyze the opportunities to provide new multimedia services. Moreover, a SDN framework is also presented to provide QoS for different multimedia services. This framework uses OpenFlow, Network Virtualization and establishes functional boxes and interfaces to test different routing algorithms. Then, the modules of "Network Performance" and "QoS Routing Algorithm" are implemented to demonstrate the effectiveness of the framework. The experiments with video streaming information show a quality optimization (PSNR, SSIM, MOS) in comparison with the best effort engine. Finally, the challenges of SDN and the future lines of work are presented.

1. Introduction

Typical network architecture was originally designed to transmit information and provide connectivity through the best-effort paradigm. The information, indistinct of the type of data, has the same treatment at the network layer. The upper layers have the task of reading the information and classifying the different services (e-mail, http, ftp). In this context, the route between source and destination is established in a distributed system, where an individual and not a global view of the network is used. Each network device uses a routing protocol to establish the network topology and the routing algorithm computes the next hop. This structure has worked properly for data transmission and facilitated the expansion of Internet in new areas of human activity.

However, the exponential growth of devices connected to the Internet and the development of new post-PC services (VoIP, video streaming, Internet of Things, cloud computing) brought with it unexpected challenges to the rigid network architecture of today. For example, real time streaming information is more sensitive to delays than e-mail services and the network equipment should categorize this different data types in lower layers. In the current networks, these

* Corresponding author. Tel.: +34633570507.
E-mail addresses: angevald@ucm.es (Á.L. Valdivieso Caraguay), jesusantoniopuente@ucm.es (J.A. Puente Fernández), javierv@ucm.es (L.J. García Villalba).

http://dx.doi.org/10.1016/j.comnet.2015.09.013
1389-1286/

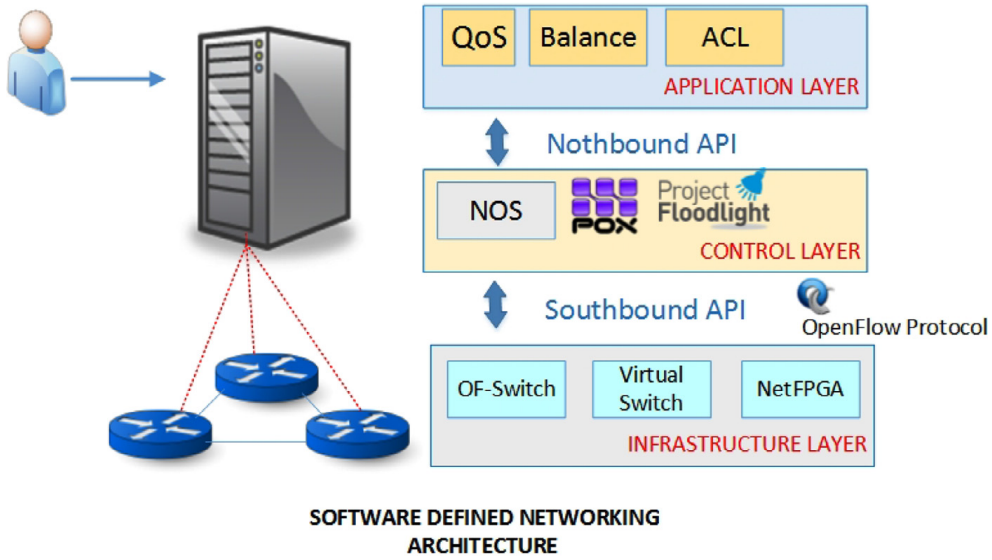


Fig. 1. Software defined networking architecture.

services have been progressively integrated in the network core through protocols (currently about 7000 RFCs) or proprietary technology. However, the updates increase the complexity of the network devices in terms of memory, CPU resources and energy consumption. Moreover, some of the new services require the redesign of the network or the change of equipment due to the rigid union between hardware and software. In addition, the administrator has limited access to the network configuration (change parameters using command line interface CLI). These limitations have also brought another requirement to provide better network services, namely the flexibility and customization of the network behavior. In other words, it is necessary that the network can be adapted to the specific needs of a particular institution, campus or group of users.

Software defined networking (SDN) is a novel paradigm that proposes the separation between data and control planes in network devices and centralizes the control of the network. SDN opens the possibility to remotely controlling the network behavior through a software program in an external server. The possibility of developing "network programs" has also given rise to the concept of Network Operating System NOS, following the evolution of operating systems OS in computer science. In this piece of work, we describe the software defined networking architecture and OpenFlow [1] as one of the main SDN protocols. This advance has been tested in different fields, such as security, home networking, data center, virtualization, Internet of Things, amongst others [2,3]. Special attention is given to the development of multimedia services, such as quality of service. In this context, the network administrator can dynamically implement different algorithms without waiting for the publication in a protocol or software/hardware update of the network device. This paper also describes the first initiatives of multimedia applications using a SDN/OpenFlow approach. Furthermore, we present a SDN framework to evaluate different QoS Routing Algorithms. We use virtualization, the OpenFlow protocol and the per-flow engine to propose functional boxes to provide different multimedia services. Moreover, we implement and propose an algorithm to monitor the network performance and select the path between source and destination. Finally, we present the results and describe the challenges and future work.

The rest of this paper is organized as follows: the software defined networking architecture together with OpenFlow protocol is presented in Section 2. Section 3 presents the related work of the first multimedia applications with SDN. The framework to evaluate QoS Routing Algorithms is presented in Section 4. Section 5 describes the network performance and QoS Routing Algorithm functions and our experiences in the implementation of the framework. Section 6 presents an application scenario, test and results. Finally, Section 7 describes the conclusions and Section 8 the future work.

2. Software defined networking

The software defined networking SDN architecture takes advantages of previous advances in network technologies. Active Networks [4] in mid 90s proposed a network interface or API to open the network resources and enabling the individual programming of the node. The administrator can program the treatment of information based on the header of the incoming packet. This approach helps researchers to test new network architectures, models and algorithms [5,6]. However, their implementation in network productions was unfeasible due to limitations in performance and safety. Another important advance in network technologies is the separation between control and data planes [7,8], enabling their independent evolution. Thus, the data plane is responsible for the packet forwarding engine (transmission of packets) and the control plane is in charge of the network management (routing engine, QoS, load balancing). However, the first implementations of this systems were difficult due to the need for custom switches based on Linux, OpenWrt or NetFPGA.

Software defined networking proposes a separation between control and data planes in network devices, a centralized control of the network and establishes open interfaces between them [9]. The SDN architecture is shown in Fig. 1. The network devices or infrastructure layer are in charge of

the forwarding of the data. For its part, the control layer reads the actual and global situation of the network and remotely controls the network behavior. The network administrator creates network applications using the functions provided by the control layer. The union between infrastructure and control layer is given through an API. One of the commonly open API is OpenFlow [10] published by the Open Networking Foundation.

OpenFlow takes the common elements (routing tables) and network functions (read, send, drop packets) available in current network devices and opens them up. This approach enables the external control of the network and facilitates their implementation. An OpenFlow switch (OF-Switch) is internally composed of flow tables that contain three fields: packet header, action and statistics. The headers of the incoming packets are compared with the packet header of the flow tables. In the case of a match, the corresponding actions are executed. Otherwise, the packet is sent to the controller. The principal actions defined by the protocol are: send the packet to a port, send the packet to the controller or drop the packet. The statistics field are counters that record statistic information for monitoring and administration purposes.

The controller remotely modifies the flow tables of the switches using the OpenFlow Protocol [10] (Southbound API). The controller uses the functionalities of the switch using a Network Operating System (NOS). The network programmer uses the basic functions of the NOS (topology, link states, port status) and creates applications and services in function with the needs of the customers. An API between control and application layer is also proposed in order to facilitate the implementation of high level network policies (Northbound API). Some of the applications based on SDN include Home Networking, Security, Mobile Networks, Reliability and Recovery, Cloud Computing, Multimedia, amongst others [2,3]. This work pays special attention to the development of SDN to provide quality of service architectures.

3. Related work

Some initiatives have already analyzed the opportunities of SDN/OpenFlow in the development of QoS applications. In [11], the authors propose a SDN Framework to adaptive video streaming enabling a dynamic QoS routing support. The framework offers two levels of QoS and best effort routing. The routes are calculated based on the Constrained Shortest Path (CSP) problem and the used algorithm is the polynomial-time LARAC Algorithm [12]. The QoS Framework presented in [13] offers a QoS API with slicing capabilities. This API extends OpenFlow switch with rate-limiters and priority queue capabilities in order to preserve network resources to a specific network slice. Furthermore, in [14] the authors propose a Network Control Layer NCL integrating OpenNaaS with SDN. The NCL has an Application, Controller and a Monitor module. For its part, the OpenNaaS contains a Platform, Resource and Remote applications. The PolicyCop [15] proposes a framework to configure Service Level Agreements (SLA) and provide QoS. This framework consists of a data, control and management plane with a Policy Validator and a Policy Enforcer component. This tool uses OpenFlow switches and controllers to guarantee SLAs to users. The work presented in [16] addresses the integration of OpenFlow with large scale WAN networks. The B4 Googles WAN uses the advantages of OpenFlow to split application flows among multiple paths, balance load and use several links at 100% capacity and all links to average 70% utilization.

Some authors consider the term quality of experience (QoE) as an evolution of quality of service (QoS). QoE takes into account the perceived acceptance of the user to a particular service. However, this term is in development and requires a solid concept. The project presented in [17] improves the quality of experience using the path optimization engine. It consists of the QoS Matching and Optimization Function (QMOF) and Path Assignment Function (PAF). The system reacts to a degradation of the links updating the paths through the network. The OpenFlow-assisted QoE Fairness Framework (QFF) [18] optimizes the QoE of video streams taking into consideration the end-device capabilities. This framework monitors multimedia streams in the network and adjusts the routes based on the available network resources and the users device-based requirements.

4. Framework for optimized QoS routing

The proposed framework consists of different functional boxes as described in Fig. 2. The global architecture includes the infrastructure, virtualization, control and application layers. The infrastructure layer represents the elements responsible for the forwarding process (switches, routers and other data plane elements). The virtualization layer [19] logically abstracts the data plane resources of the infrastructure layer. Then, the service provider establishes different "slices" of logical topologies and assigns each slice to the different users. Flowvisor [20] is an example of network virtualization in SDN networks. Similar to virtualization in computer systems, this virtualization layer is transparent for the upper layers. For this reason, the control layer can use OpenFlow protocol as Southbound API. The functional boxes of the control layer are based in the "per flow" separation of traffic and the information provided by the OpenFlow Protocol. Additionally, the framework uses Rest as Northbound API to communicate with the application layer. The network programmer creates network applications based on the functions provided by the control plane.

The descriptions of the application layer functional boxes are defined as follows:

- QoS parameters. This module configures the different parameters to the QoS routing module. These parameters can include data types, priorities, QoS levels, devices, amongst others. These values are sent to the QoS Routing Algorithm through a well-known northbound API (RestAPI).
- Inter-controller parameters. The user establishes the parameters and the information to be sent to another network or external SDN controller.

The descriptions of the control layer functional boxes are defined as follows:

- Topology Manager. Recognize the network devices (switches, links) present in the infrastructure layer and their capabilities. Topology Manager uses the information of OpenFlow messages and LLDP (Link Layer Discovery

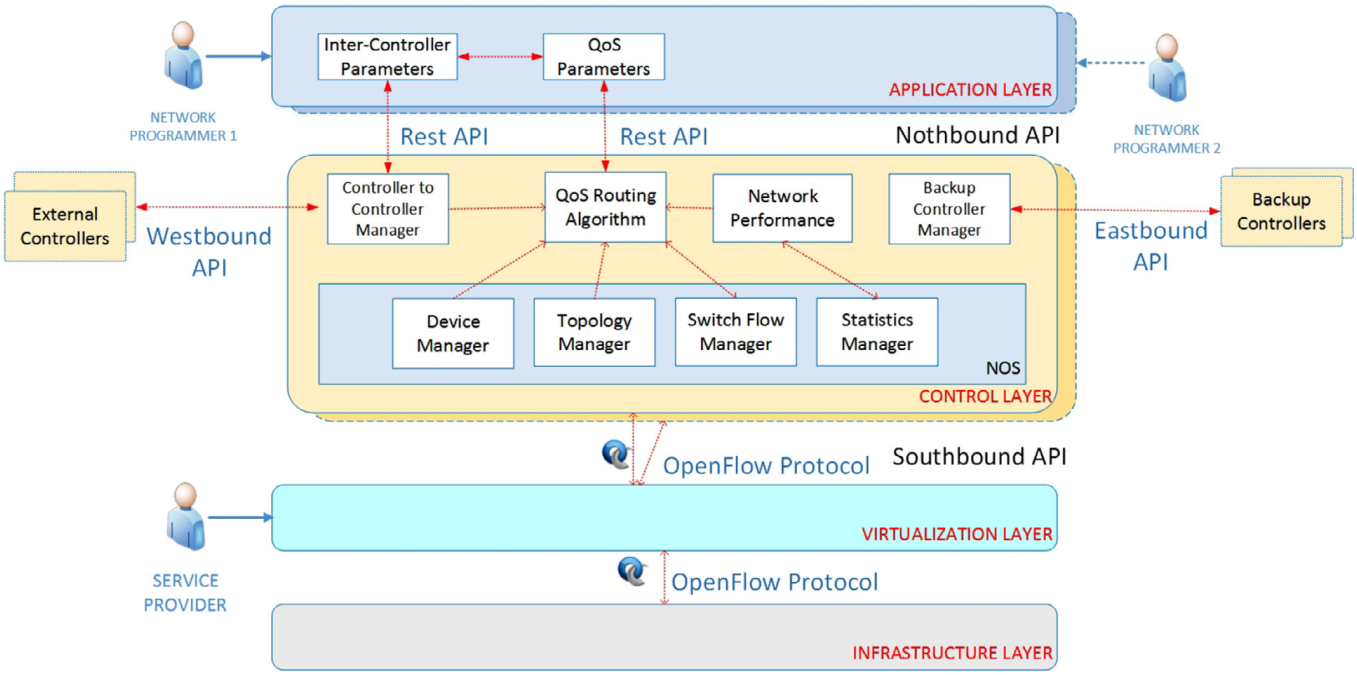


Fig. 2. QoS proposed framework.

Protocol) to infer the topology. The Topology is represented as a graph $G(N, A)$ where N is the set of switches and A the set of arcs (links). This information is available to other functions and modules of the controller.

- Device Manager. Identifies and tracks physical hosts or terminal devices (server, client, mobile phone, webcam, sensor) and their location in the network. The location of a device within the network is saved as the switch id and port id where it is connected $(Ni, pi) \in G$. Device Manager Service can find the position of a device based on his MAC or IP address and send this information to other modules.
- Controller to Controller Manager. Establish the communication with other controllers through a westbound API. This API can be used to send information to other networks (e.g. Autonomous System AS). This module can learn the existence of new hosts or final devices connected externally. This information is sent to the QoS Routing Algorithm.
- Backup Controller Manager. Periodically send updates to backup controllers. In case of Denial of Service attack or a Network Operating System NOS failure, the backup controller can automatically take control of the whole network.
- Statistics Manager. Read and organize the statistical information sent by the counters of the switches. This information can be analyzed to detect failures or congestion of links.
- Switch Flow Manager. Receive the paths and priorities calculated by the QoS Routing Algorithm and configure the flow tables of the switches.
- Network performance. Uses the information provided by the statistics manager and execute an algorithm to evaluate the performance of the network. Different performance algorithms can be implemented in order to improve the accuracy of the results. The variables can include bandwidth, data rate (bytes or packets per second), packet loss rate, delay, among others. This information is continuously being sent to the QoS Routing Algorithm for the path calculation.
- QoS Routing Algorithm. Read the QoS parameters provided by the network administrator and calculate the appropriate paths for the information circulating through the network. This algorithm reads the topology, the devices connected and also the actual performance of the links. This means that the controller can also automatically react in case of a failure of degradation in the quality of the network. Similarly the network performance engine, this algorithm can be improved due to the other functional boxes working independently.

5. Implementation

This sections explains the engine used in the implementation of the framework. Since this work is concentrated in the SDN-quality of service approach, we describe the aspects related with network performance and QoS routing.

5.1. Network Operating System

There are different Network Operating Systems NOS (NOX/POX [21], Beacon [22], Floodlight[23], Maestro [24]), each with its individual characteristics. We selected the Floodlight controller to implement this design. Floodlight is a Java-based controller under Apache License and widely supported by a community of developers. It supports OpenFlow 1.0 and offers a modular programming environment. Moreover, it has already implemented basic modules (topology manager, device manager, statistics manager) and provides a Java API to the development of internal high speed applications together with a RestAPI interface to remote programming (e.g. python-based applications). In this

Algorithm 1: Network performance function.

Input: network graph $G(N, A)$
period monitoring time t
link bandwidth bw_{ij} for each $arc(i, j) \in A$
adjustment factor $\propto$
Result: cost c_{ij} for each $arc(i, j) \in A$

1 **procedure** CostFunction(I)
2 Start timer k with period t;
3 **foreach** *period* $k = 0, 1, 2, 3, \ldots \in t$ **do**
4 **foreach** $arc(i, j)$ **do**
5 Read the sent bytes s_i of the port of the start link with s_i^k=tx_bytes in $ofp\_port\_stats(node,port(i))$;
6 Read the received bytes r_j of the final link with r_j^k=rx_bytes in $ofp\_port\_stats(node,port(j))$;
7 **if** $k = 0$ *(initial time)* **then**
8 c_{ij}=1;
9 **if** $k > 0$ *(each period)* **then**
10 Calculate the data rate of the link with $dr_{ij}=\frac{s_i^k - s_i^{k-1}}{t}$;
11 Calculate the packet loss of the link with $pl_{ij}=\frac{(s_i^k - s_i^{k-1})-(r_j^k - r_j^{k-1})}{t}$;
12 Calculate the cost of the link with $c_{ij}=1+\frac{\propto . dr_{ij}+(1-\propto).pl_{ij}}{bw_{ij}}$;
13 **end procedure**

implementation, the basic modules are adapted to facilitate the integration and the requirements set by other modules.

5.2. Topology abstraction

The topology of switches and links present in the infrastructure plane is represented as a graph $G(N, A)$, where N is the set of switches and A the set of *arcs* (links). For instance, $arc(i, j) \in A$ represents a link from node i to node j. Each link is associated with a weight value or link cost. In this case, the symbol c_{ij} represents the link cost of the $arc(i,j)$. The link cost can represent a fixed value (hop count ($c_{ij} = 1$)) or a time-varying field, such as jitter, loss or available bandwidth. R_{sd} represents the set of paths or routes available from the source s to destination d. In this implementation, we impose several assumptions [25]: the network is directed, the network contains a directed path from node s to every other node in the network and the network does not contain a negative cycle.

5.3. Network performance

The active measurement of the performance of the network is an open challenge. The statistics manager module provided by the NOS can read the theoretical maximal data rate of a particular port through the OFP_PORT _FEATURES parameter. However, the real capacity or maximal data rate depends on the conditions and quality of the link between switches. Furthermore, the OpenFlow 1.0 specifies counters in the switch at different levels (table, port, flow, and queue) but does not provide packet delays or jitter measurements. Moreover, the controller reads this information with the inevitable time delay of the link controller-switch. The models proposed in [26,27] present algorithms to improve the performance diagnosis for OpenFlow networks. Another solution is the integration of data plane measurements or specialized tools such as sFlow [28]. In our model, the cost of the network performance module uses the engine of Algorithm 1. First, the module initializes the cost function with the value of 1, assuming that the links do not have a loss rate. Then, the module initiates a timer to continuously monitor the port statistics counters (OFP_PORT_STATS) and periodically collects the sent and received bytes in the ports of the switch. Using the information of the topology (network graph) and the information of the counters, the algorithm establishes the data rate and packet loss rate. The network performance function considers the ideal condition $s^k > s^{k-1}$ and $r^k > r^{k-1}$. The analysis of additional conditions remains as future challenges. The cost function uses this information and an adjustment factor $\propto$ to calculate the dynamic cost of the link. This value is continuously updated and sent to the QoS Routing Algorithm.

5.4. QoS Routing Algorithm

There are multiple techniques to provide QoS in typical hop-by-hop network architectures [29]. However, the global vision of the network provided by SDN and the "per-flow" OpenFlow engine can facilitate new models of efficient

Algorithm 2: QoS routing function.

Input: network graph $G(N, A)$
link cost c_{ij} for each $arc(i, j) \in A$
parameters to identify QoS packets w/QoS
(e.g. UDP port, IP src, IP dst, MAC src, MAC dst)
received OpenFlow PACKET IN messages p
Result: Set of OF-flowmod messages F to configure switches

1 **procedure** QoS Route
2 Read the header of the PACKET IN message p with h_p=$read\_header(p)$;
3 Check if h_p is part of w/QoS (e.g. UDP Port = 5532) with
if $h_p \in w/QoS$ **then**
4 Read the OF-Switch s_s from where p comes with s_s=$read\_sw\_source(p)$;
5 Read the destination IP address d_{IP} from h_p with d_{IP}=$read\_dst\_ip(h_p)$;
6 Use Device Manager Service to find OF-Switch s_d where d_{IP} is connected with s_d=$DM\_find\_device\_ip(d_{IP})$;
7 Find route r between source s_s and destination s_d using links cost c_{ij} with r_{sd}=$Dijkstra(G,s_s,s_d,c_{ij})$;
8 **foreach** *OF-Switch* $s_k \in r_{sd}$ **do**
9 Create OF-flowmod F_{sk} for OF-Switch s_k with high priority with
F_{sk}=$create\_flowmod\_message(s_k)$;
$F_{sk}.set\_high\_priority()$; ;
else
10 Process packet p with best effort engine and low priority with
F_{be}=$fdl\_irouting\_service(p)$;
$F_{be}.set\_normal\_priority()$;
11 **end procedure**

Routing Algorithms. In this work, we present a basic solution for QoS Routing based on a graph of nodes and links $G(N, A)$. This procedure is summarized by Algorithm 2. In this implementation, the QoS Routing module can assign different packet fields as QoS parameters (w/QoS), for example TCP or UDP port, IP source, IP destination, MAC source, MAC destination, among others. The user provides this information to the Routing module through the RestAPI. For example, the user can establish high priority (w/QoS) to video streaming that use a specific port number (e.g. UDP Port = 5532) and best effort engine and normal priority (w/oQoS) to other users.

The module also receives the link cost values c_{ij} of the network performance module and the graph $G(N, A)$ of the topology module. When the controller receives a PACKET IN p message, it analyzes the packet header h_p and establishes if the packet fulfills the QoS requirements w/QoS. In the example, the controller verifies if the UDP port number is 5532. If so, the module reads the switch s_s from where p comes and the IP destination d_{IP} in order to locate the position of destination s_d using Device Manager module. Then, this implementation uses the Dijisktra algorithm [29] to establish the route r_{sd} of the network in function with the link costs provided c_{ij} by the network performance module based on data rate and loss rate of the links. The chosen route will be the path with the dynamic minimum cost. This path is then installed in the network devices through the Switch Flow manager. The Switch Flow manager creates F OpenFlow flowmod messages to configure the flow tables of the switches and assign (w/QoS) packets with higher priority. In case of (w/oQoS) traffic, the Floodlight IRouting service uses the minimun hop count engine to establish a path between source and destination with normal priority.

6. Application scenario

The framework is tested using the topology shown in the Fig. 3. Also, the test method described in [12,30] has been taken into account as baseline, adapted and expanded

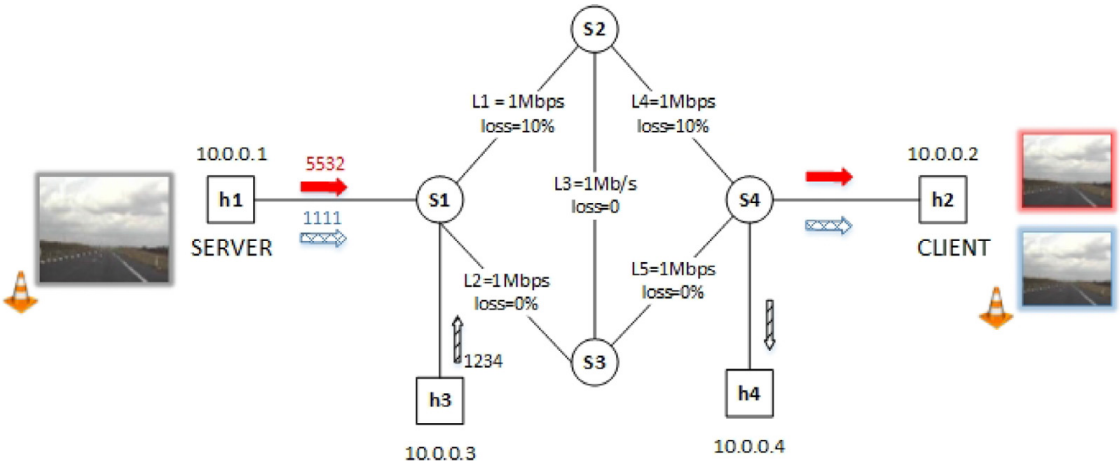


Fig. 3. Test topology.

to include new issues and metrics, as detailed below. The topology is emulated using a server (Intel core i5 2.4 Ghz, 4GB DDR3 RAM, Mac Osx v10.9.4) with a VM Linux Ubuntu 13.04 (64 bits, 512Mb RAM, 1 CPUs, 8GB HD). The virtual machine executes the open source Mininet emulation tool 2.1.0 [31,32]. Mininet enables the creation of custom topologies of OF-enabled switches, links and virtual hosts using Python scripts. Despite the fact that Mininet is currently one of the most used platforms to perform SDN experiments, the switching capacity in real time applications is limited by underlying host system capacity [33]. Limited topologies with slower links provide and guarantee accuracy in network emulation in terms of timing accuracy and CPU/memory isolation.

The test topology has four hosts (h1–h4), four switches (S1–S4) and five links between them (L1–L5). Two links (L1, L4) are assigned with a loss of 10%. Mininet uses a netem linux kernel component (part of iproute2 package) to emulate an independent loss probability on the virtual links. In other words, 10% of the packets moving through links (S1 -> S2 & S2 -> S3) are randomly dropped.

This application uses the video file "highway_cif" [34] and the RTP/UDP streaming protocol using VLC server. The video is encoded in MPEG-1/2 (highway_cif.ts) with a data size of 2.6 MB (2 572 968 bytes), resolution of 352 × 288, 2000 frames and a duration of 80 s using the ffmpeg tool. The average bitrate of the video file is 0.25 mbit/s. However, the ethernet frame and packet headers, RTP control messages and headers and additional information increases the real data rate of the network links.

In our experiments, the real data rate of a simple video stream is increased by about 0.4 mbit/s. For this reason, we limited the links to 1 Mbps and send different flows of videos in order to saturate the network.

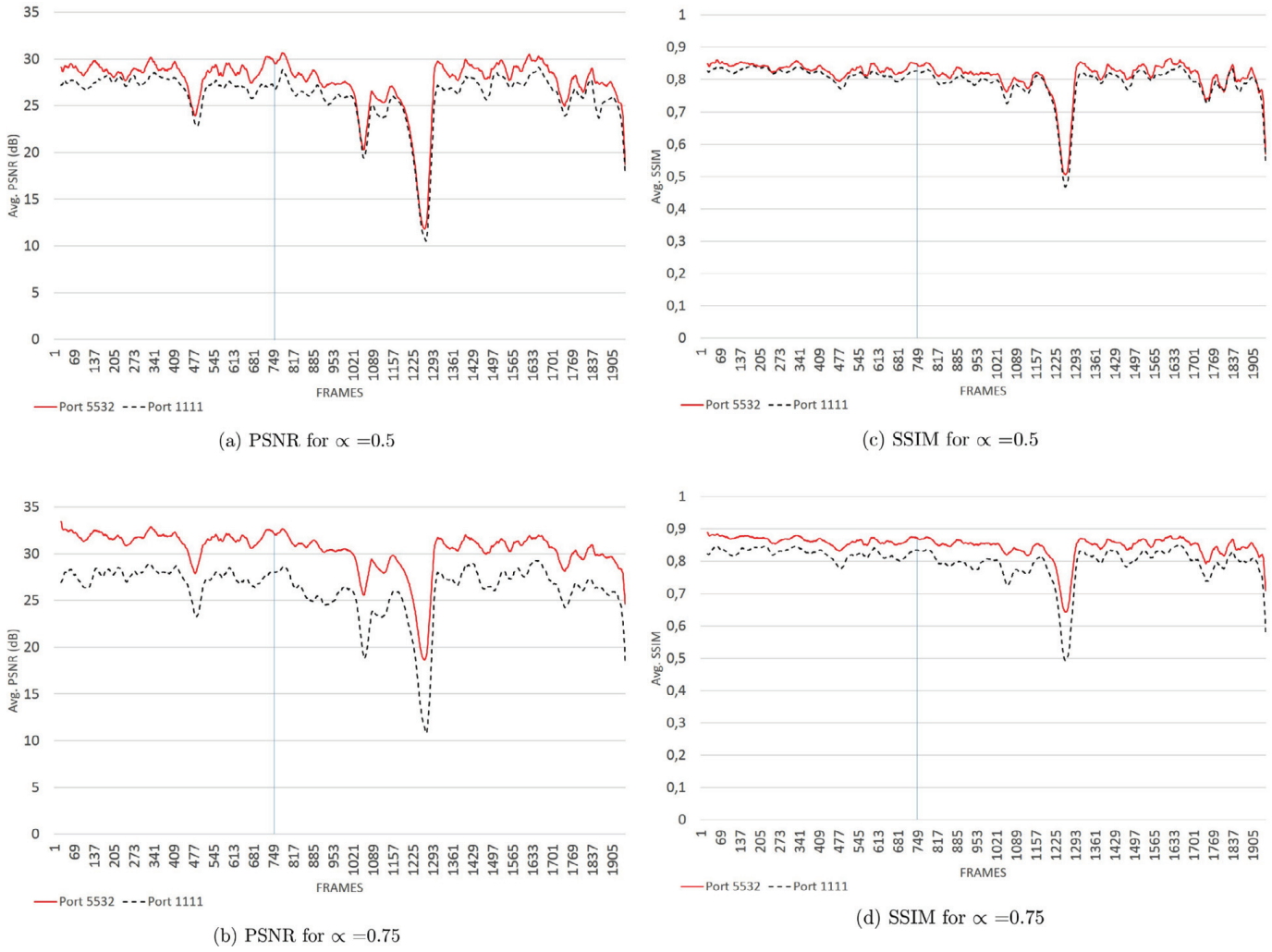
The controller behavior is highly dependent on the selected NOS (Floodlight, NOX, Beacon, Opendaylight), each with its own characteristics. The NOS performance depends on the programming language (C++, Java, Python), the equipment, work load, topology and forwarding complexity [35]. In [2], we have presented an analysis of the performance of the different SDN controllers. The selected NOS is Floodlight version 0.90 that is also executed in the same VM. The performance of the controller is measured using Cbench tool [36] in terms of throughput and latency as described in Table 1. The first row describes the Floodlight processing time of a switch request. In this case, the switch sends a request (82 byte size packet in) and waits to the corresponding controller response. For its part, the second row shows the requests rate that a controller can handle. For that, the switch attempts to saturate the controller sending as many multi outstanding requests as buffering allows.

Table 1
Performance analysis of the Floodlight controller.

Measure	Min.	Max.	Avg.	Std. dev.
Latency (ms)	0.161	0.895	0.315	0.604
Throughput (request/s)	421.33	21975.66	13558.46	7432.06

In the experiment scenario, assuming that all the switches have an empty flow table and send all received PACKET IN messages to the controller (worst case scenario), the total requests number is about 914 request /second. This corresponds to the 7% of the average Floodlight capacity. As a result, the experiment follows the recommendations to minimize mutual interference between the processes (Mininet, Floodlight, VLC server and client, ffmpeg) and maximize the accuracy of the results.

The experiment uses three flows: the high priority flow, a normal priority flow, and a background flow. The high priority flow uses the port 5532 to send the stream between h1 and h2 with high priority (w/QoS). For its part, the normal priority flow (w/oQoS) uses the port 1111 to send the stream. The background flow sends the stream between h3 and h4 to saturate the links. During each run, 2000 frames were sent to each flow with an average rate of 25 frames/s. The background flow was introduced after 750 frames ($t = 30$ s). The samples were periodically monitored every 125 frames (5 s). The streams are saved in different video files. Then, the files are decompressed as .yuv files and the PSNR (Peak Signal to Noise Ratio) and SSIM (Structural Similarity Index Metric) between source and destination is calculated using the Evalvid tool [37,38]. The PSNR measures the ratio between the original and the error signals of the video, while SSIM analyzes the perceptual distortion of the two fonts.



(a) PSNR for $\propto$ =0.5

(c) SSIM for $\propto$ =0.5

(b) PSNR for $\propto$ =0.75

(d) SSIM for $\propto$ =0.75

Fig. 4. Results of PSNR and SSIM: (a) and (c) for $\alpha = 0.5$, (b) and (d) for $\alpha = 0.75$.

Mininet as network emulator (data plane) uses the linux kernel and lightweight virtualization to emulate and run virtual host, links and software-based network elements with a behavior similar to discrete hardware elements. Similarly, the Floodlight controller is an SDN Controller that works with physical and virtual OpenFlow enabled switches. In this experiment, the Floodlight app and mininet are separate processes which use OpenFlow (TCP port 6633) to interchange information between them (as a real network). Furthermore, the VLC streaming server and client are executed in separated processes running in virtual hosts (in the same VM). This causes slight variations between tests depending of the available CPU and memory resources. For this reason, we perform a basic Monte-Carlo method, that is, repeat the test scenario several times (20) and evaluate the corresponding average. However, in our experiments, the results of a single test present the same pattern, trends and similar conclusions.

The controller will assign different paths to the flows, even though they have identical (src ip, dst ip) tuples. In the first case (w/o QoS), the route is chosen based only in the minimum hop count. In the second case (w QoS), the controller identifies the load and loss in the different links. With this information, the controller intends to find the best route to the flow.

The algorithm assigns the path S1->S2->S4 as (w/o QoS). For its part, the traffic assigned as (w/QoS) avoids load and loss choosing the path S1->S3->S4. Furthermore, the controller configures the switches to process (w/QoS) packets with high priority.

The accuracy of measured values by the network performance module depends on the monitoring interval. However, it is also verified that a reduction of period increases the CPU load of floodlight process and the number of requests on the switches. The network performance module is configured with a period monitoring time of 5 s.

In order to reduce distortions, we display the trend line with an average of 25 frames. The results of the experiments are depicted in Fig. 4. The red line (solid) represents the high priority flow (w/QoS) and the black line (dotted line) represents the normal priority flow (w/o QoS). Fig. 4a and c shows the PSNR and SSIM against the number of frames with a factor adjustment of $\propto$= 0.5 (worst case). For its part, Fig. 4b and d are the results for a factor adjustment of $\alpha = 0.75$ (best case). As expected, during the transmission period, the traffic with QoS clearly show better levels of efficiency compared with the w/o QoS traffic. For the case of $\alpha = 0.5$, the average PSNR for w/ QoS is 27.47 dB against 26.00 dB of w/o QoS and the average SSIM is 0.81 against 0.79 accordingly. For

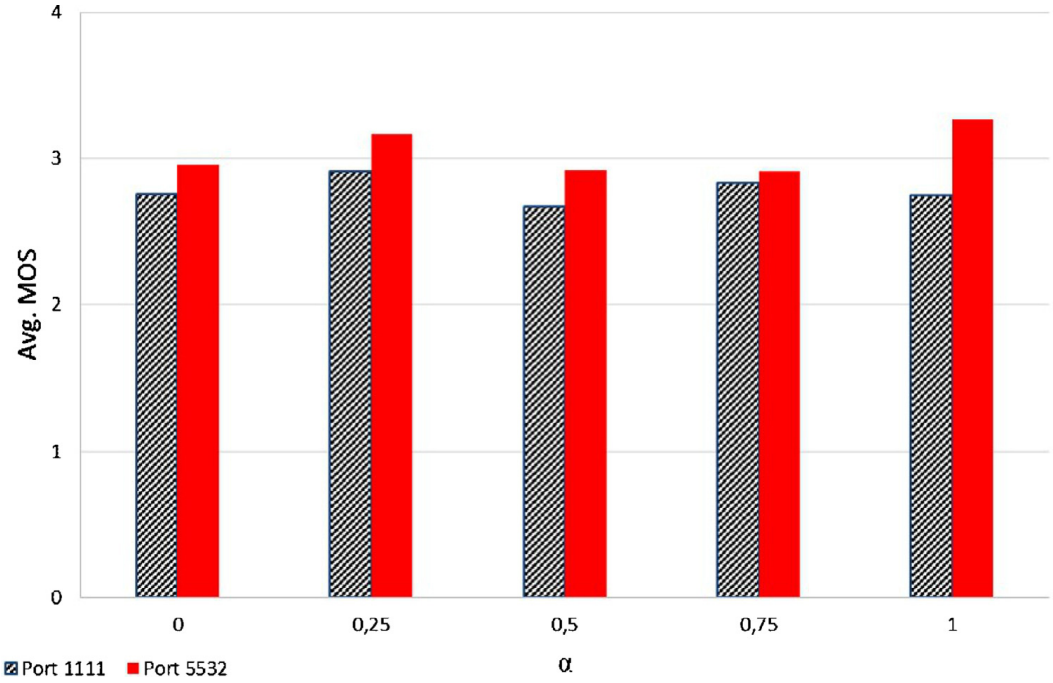


Fig. 5. Results of MOS for the different values of α.

the case of $\alpha = 0.75$, the PSNR value for w/QoS is 30.41 dB against 26.18 dB of w/o QoS and the SSIM 0.85 against 0.80. However, Fig. 4 presents several troughs in case plot. This effect is mainly caused by the fast moving objects and dynamic events in video scenes. During troughs, the video has an increase in bits per frame and data rate up to 0.302 mbit/s (increase of 21% of average video content bitrate). This increase, in turn, causes major congestion in links and reduces the measured quality perception.

In order to summarize the results of the experiments, the MOS (Mean Opinion Score) is taken into account. The difference with respect to the factors mentioned above (PSNR, SSIM) is that the MOS (ITU recommendation P.800 [39]) represents a subjective value of the users perceptions with respect to a service.

In other words, the MOS represents a quality of experience (QoE) metric which estimates the grade of acceptability of the user. It proposes a scale of perception of 5 points: (5) excellent, (4) good, (3) fair, (2) poor and (1) bad. The procedure to calculate the MOS is based on relation scales with QoS metrics and is explained in [38]. Fig. 5 represents the average MOS of the two streaming in function of α. In this analysis, the perception of w/QoS streaming is in all cases better as w/o QoS and on average near or above 3 (fair).

We have performed additional tests using similar topology and sending only the high-priority flow. Even under excellent network conditions, the reception of the real time video streaming suffers variations caused by jitter, video decoder algorithm, CPU and memory resources of network and terminal device, among others. The obtained average MOS is 3.2 similar to the high priority MOS values of the experiments with additional flows (normal priority flow and background flow). This behavior demonstrates that the controller ensures QoS of high priority flow and balances other flows in the remaining available resources.

7. Conclusions

This paper presents an introduction to the innovative paradigm software defined networking SDN. This work also describes relevant initiatives of the SDN / OpenFlow solutions for multimedia applications. Moreover, an SDN architecture is proposed for the development and testing of different QoS Routing Algorithms based on OpenFlow Protocol v 1.0. This work has also analyzed and tested the limitations of the Floodlight controller. The results of the experiments using an emulation tool and a sample topology demonstrate the advantages of our framework. The results in terms of PSNR, SSIM and MOS for a RTP/UDP streaming service demonstrate that the SDN controller ensures quality of service for a high priority real streaming flow in comparison with a typical best effort engine.

8. Future work

Therefore, the challenges for future research include the improvement of network performance and QoS Routing Algorithms. The development of experiments with high bandwidth links, large number of flows and low loss rates in real environments and different NOS controllers remains for future work. The algorithms can use global centralized control to optimize the flow placement and traffic engineering techniques in LAN/WAN networks. The customized SDN network can organize high-priority flows along lossless paths and low-priority flows along contended paths. Similarly, multipath forwarding and tunneling can optimize the network resources as a function of application priority. Also, the appropriate monitoring solution for SDN architectures is an open challenge. Valuable information such as delay, jitter, bandwidth, connection establishment time, packet loss and throughput is not directly provided by the OpenFlow

protocol. The information provided by OpenFlow requires further mining and analysis. The measurement of these variables could be implemented on the data plane (more expensive), control plane (less efficient) or both. Moreover, the inclusion of custom data plane implementations (traffic classification, net-FPGA implementations, encryption, or packet processors) within SDN can be analyzed. The coordination between data and control planes to avoid overheads in network hardware and controller as well as the coexistence and integration between traditional and SDN networks is an open challenge.

For its part, the security in SDN is a critical key aspect. SDN networks should be secure against external malicious attacks (DDoS) or internal failures (NOS failure or incorrect programming) in order to be implemented in production networks. For this purpose, the separation of group profiles with different access policies and the design of early attack detection strategies are desirable. Finally, it is necessary to have an evolution of new generation of Service Levels Agreement SLAs as well as the coordination between service providers, industry and researches.

Acknowledgments

The research leading to these results has been partially funded by the European Unions H2020 Program under the project SELFNET (671672). Ángel Leonardo Valdivieso Caraguay is supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program No. 2543-2012.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[2] A.L. Valdivieso Caraguay, L.I. Barona López, L.J. García Villalba, Evolution and challenges of software defined networking, in: Proceedings of the 2013 IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1–7.
[3] A.L. Valdivieso Caraguay, L.I. Barona López, A. Benito Peral, L.J. García Villalba, SDN: evolution and opportunities in the development IoT applications, Int. J. Distrib. Sens. Netw. 2014 (2014).
[4] K. Calvert, et al., Architectural Framework for Active Networks Version 1.0, Active Networks Working Group Draft (1999).
[5] A. Galis, B. Plattner, J.M. Smith, S. Denazis, E. Moeller, H. Guo, C. Klein, J. Serrat, J. Laarhuis, G.T. Karetsos, et al., A flexible IP active networks architecture, in: Active Networks, Springer, 2000, pp. 1–15.
[6] B. Schwartz, A.W. Jackson, W.T. Strayer, W. Zhou, R.D. Rockwell, C. Partridge, Smart packets for active networks, in: Proceedings of the 1999 IEEE Second Conference on Open Architectures and Network Programming Proceedings, IEEE, 1999, pp. 90–97.
[7] L. Yang, R. Dantu, T. Anderson, R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, (RFC 3746 (Informational)) April 2004.
[8] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, Design and implementation of a routing control platform, in: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, NSDI'05 - Volume 2, USENIX Association, Berkeley, CA, USA, 2005, pp. 15–28.
[9] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in SDN-OpenFlow networks, Comput. Netw. 71 (2014) 1–30.
[10] OpenFlow Switch Specification v.1.0.0 1-42, 2009.
[11] H.E. Egilmez, S. Civanlar, A.M. Tekalp, An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks, IEEE Trans. Multimed. 15 (3) (2013) 710–715.
[12] A. Juttner, B. Szviatovski, I. Mécs, Z. Rajkó, Lagrange relaxation based method for the QoS routing problem, in: Proceedings of the IEEE Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2001, 2, IEEE, 2001, pp. 859–868.
[13] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, P. Yalagandula, Automated and scalable QoS control for network convergence, Proc. INM/WREN 10 (2010) 1.
[14] I. Bueno, J.I. Aznar, E. Escalona, J. Ferrer, J.A. Garcia-Espin, An OpenNaaS based SDN framework for dynamic QoS control, in: Proceedings of 2013 IEEE SDN for Future Networks and Services (SDN4FNS), IEEE, 2013, pp. 1–7.
[15] M. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, et al., PolicyCop: an autonomic QoS policy enforcement framework for software defined networks, in: Proceedings of the 2013 IEEE SDN for Future Networks and Services (SDN4FNS), IEEE, 2013, pp. 1–7.
[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: experience with a globally-deployed software defined WAN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.
[17] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, P. Dely, Towards QoE-driven multimedia service negotiation and path optimization with software defined networking, in: Proceedings of the 2012 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), IEEE, 2012, pp. 1–5.
[18] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, N. Race, Towards network-wide QoE fairness using openflow-assisted adaptive video streaming, in: Proceedings of the 2013 ACM SIGCOMM workshop on Future Human-centric Multimedia Networking, FhMN '13, ACM, New York, NY, USA, 2013, pp. 15–20.
[19] N. Chowdhury, R. Boutaba, A survey of network virtualization, Comput. Netw. 54 (5) (2010) 862–876.
[20] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, Flowvisor: A Network Virtualization Layer, Technical report, 2009. OpenFlow Switch Consortium.
[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2008) 105–110.
[22] D. Erickson, The beacon openflow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking, ACM, New York, NY, USA, 2013, pp. 13–18.
[23] Floodlight. (http://www.projectfloodlight.org/).
[24] Z. Cai, A.L. Cox, T.E.N. Maestro, Maestro: A System for Scalable OpenFlow control, Technical report TR10-08, 2010. Rice University.
[25] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
[26] M. Karl, J. Gruen, T. Herfet, Multimedia optimized routing in openflow networks, in: Proceedings of the 2013 19th IEEE International Conference on Networks (ICON), IEEE, 2013, pp. 1–6.
[27] M. Shibuya, A. Tachibana, T. Hasegawa, Efficient performance diagnosis in openflow networks based on active measurements, in: Proceedings of the Thirteenth International Conference on Networks ICN 2014, 2014, pp. 268–273.
[28] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, V. Maglaris, Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments, Comput. Netw. (2013) 122–136.
[29] P. Van Mieghem, F.A. Kuipers, Concepts of exact QoS routing algorithms, IEEE/ACM Trans. Netw. 12 (5) (2004) 851–864.
[30] H. Egilmez, S. Dane, K. Bagci, A. Tekalp, OpenQoS: an openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks, in: Proceedings of 2012 Asia-Pacific Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012, pp. 1–8.
[31] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 2010, pp. 1–6.
[32] M. Gupta, J. Sommers, P. Barford, Fast, accurate simulation for SDN prototyping, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking, ACM, 2013, pp. 31–36.
[33] Mininet. (https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#limits).
[34] Highway. (http://www2.tkn.tu-berlin.de/research/evalvid/cif/highway_cif.264).
[35] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, USENIX Association, Berkeley, CA, USA, 2012, p. 10.

[36] Cbench. (https://github.com/andi-bigswitch/oflops/tree/master/cbench).
[37] Evalvid. (http://www.tkn.tu-berlin.de/menue/research/evalvid/).
[38] J. Klaue, B. Rathke, A. Wolisz, Evalvid–a framework for video transmission and quality evaluation, in: Computer Performance Evaluation. Modelling Techniques and Tools, Springer, 2003, pp. 255–272.
[39] ITU-T P.800. Methods for Subjective Determination of Transmission Quality - Series P: Telephone Transmission Quality; Methods for Objective and Subjective Assessment of Quality, 1–37(1996).

Ángel Leonardo Valdivieso Caraguay was born in Loja, Ecuador, in 1985. He received a B.S. degree in Electronics and Telecommunications Engineering from the Escuela Politcnica Nacional, Quito, Ecuador in 2009 and a M.S. degree in Information Technology from the University of Applied Sciences Hochschule Mannheim, Germany in 2012. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His research interests include computer networks, information security, software-defined networking and network functions virtualization.

Jesús Antonio Puente Fernández was born in Madrid, Spain in 1988. He received his Computer Science Engineering degree in 2012 and a M.S. degree in Computer Science from the Universidad Complutense of Madrid, Spain in 2014. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His main research fields are computer networks and software-defined networking.

Luis Javier García Villalba received a Telecommunication Engineering degree from the Universidad de Málaga (Spain) in 1993 and holds a M.Sc. in Computer Networks (1996) and a Ph.D. in Computer Science (1999), both from the Universidad Politécnica de Madrid (Spain). Visiting Scholar at COSIC (Computer Security and Industrial Cryptography, Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium) in 2000 and Visiting Scientist at IBM Research Division (IBM Almaden Research Center, San Jose, CA, USA) in 2001 and 2002, he is currently Associate Professor of the Department of Software Engineering and Artificial Intelligence at the Universidad Complutense de Madrid (UCM) and Head of Complutense Research Group GASS (Group of Analysis, Security and Systems) which is located in the Faculty of Computer Science and Engineering at the UCM Campus. His professional experience includes projects with Hitachi, IBM, Nokia Safelayer Secure Communications and H2020 projects. His main research interests are computer networks, information security, and software-defined networking (SDN).

The 7th International Conference on Information Technology

Big Data

ISSN 2306-6105

ISBN 978-9957-8583-3-9

Conference Proceeding

Full

Prepared and Edited by:

- ICIT15 General Chair
  - Ali Al-Dahoud, Dean of Science and IT Faculty
- Editorial Board
  - Hani Mimi, ICIT15 Co-chair
  - Khalid Jaber, ICIT5 Co-chair
  - Israa Sabatin, Designer
  - Hanade Al-Shawabkeh, Editor
  - Ayman Al-Qafa'an, Editor

The Hashemite Kingdom of Jordan

The Deposit Number at the National Library

(2015/4/1450)

009

نسخة / مركز الايداع

Information Technology (7:Amman:2015)

The 7th International Conference on Information Technology / Ali As'ad Al-Dahoud. – Amman: Al-Zaytoonah University of Jordan, 2015

(163) p.

Deposit No. : 2015/4/1450

Descriptions: /Computers //Conferences//Information Technology/

ICIT 2015 The 7th International Conference on Information Technology
doi:10.15849/icit.2015.0038

An Overview of Integration of Mobile Infrastructure with SDN/NFV Networks

Ángel Leonardo Valdivieso Caraguay, Lorena Isabel Barona López, Luis Javier García Villalba
Group of Analysis, Security and Systems (GASS)
Department of Software Engineering and Artificial Intelligence (DISIA)
Faculty of Information Technology and Computer Science, Office 431
Universidad Complutense de Madrid (UCM)
Calle Profesor José García Santesmases, 9
Ciudad Universitaria, 28040 Madrid, Spain
Email: {angevald, lorebaro}@ucm.es, javiergv@fdi.ucm.es

***Abstract*— The growing number of on-line applications and services running on wireless and mobile devices has been limited by the rigidity of actual IT infrastructure, in which the closed union between data and control planes limits the possibility of customize the network behavior. In this context, the concepts of SDN and NFV appear as a viable solution to open the infrastructure to developers in order to create new services and applications. In this work, we describe the concepts of SDN, NFV and analyze the possibility of integrate these technologies in mobile networks. Furthermore, we present the last projects and an architecture proposal focused on this direction. Finally, we discuss the trends and challenges in order to implement these advances in production networks.**

Keywords—Mobile Network; Network Function Virtualization; OpenFlow; Software Defined Networking.

I. INTRODUCTION

The diversity of network infrastructures has enabled the increase of connectivity among users and consequently it has promoted the establishment of new business models. This new digital environment requires an IT infrastructure capable to ensure high level of Quality of Service (QoS) and customization of applications. However, the heterogeneity of cellular and wireless technologies and the current configuration techniques complicate the control and management of the network.

The IT infrastructure is composed by a set of hardware devices running proprietary software that analyzes the traffic and selects the optimal route to the destination. In this scenario, the network administrator does not have access to modify the internal operation of the device. Instead, the administrator can only configure a minimum set of parameters to modify the network behavior. Moreover, the inclusion of new services requires the individual updating of devices or the complete replacement of hardware infrastructure. For this reasons, the idea of separate the data plane and control plane in order to customize the network behavior has gained importance. Similarly, the possibility of encapsulate the different network functions based on actual network conditions can optimize the allocation of available resources.

The concepts of Software Defined Networking (SDN) and Network Function Virtualization (NFV) have changed the vision of typical network infrastructure. SDN separates the data and control planes in network devices and establishes a centralized control of the network behavior. This architecture enables to the network administrator the possibility to design and develop "network applications" and dynamically control the network. For its part, NFV allows the deployment of virtualized network functions (e.g. load balancers, firewalls) as virtual instances over standardized hardware (storage, network and servers). This technology integrates the use of different resources (servers, storage, IT-hardware), enhances the scalability of the network services and reduces the capital and operational cost.

A techno-economic analysis in mobile infrastructure reveals that the benefits of the introduction of SDN and virtualization techniques could decrease the capital expenditures. The capital expenditure could be reduced around 13.81 % in a SDN scenario [1]. It is clear that architectures based on SDN offer multiple potential advantages for telecom operators [2], for instance, the possibility of deploy Radio Base Stations in the Cloud [3] or integrating LTE network elements with SDN switches managed from the cloud [4]. In this context, the industry and research community go a step further in this direction and have been combining their efforts in multiple projects such as OpenRoads [5], SoftCell [6] as well as European Projects such as T-NOVA [7], UNIFY [8] among others. In this piece of work, we describe the concept and the evolution of SDN and NFV in the last years. Furthermore, the integration of mobile infrastructure with SDN/NFV as well as the trend and challenges to implement these technologies in production networks are analyzed.

The work is structured as follows: in Chapter II the concepts of SDN and NFV are presented. Next, Chapter III reviews the integration of mobile networks with both

doi:10.15849/icit.2015.0038

technologies. Chapter IV analyzes the trends and challenges and provides an initial SD/NFV architecture. Finally, Chapter V presents the conclusions.

II. SDN/NFV

A typical network device is composed by an integrated data plane and control plane. The data plane receives the packet, reads the header information, sends the information to the control plane and forwards the packet to the next network device. For its part, the control plane analyzes the information provided by the data plane and executes a routing algorithm to establish the optimal route to the destination. Once the route is chosen, the control plane sends the decision to the data plane. However, the limited coordination and access to the configuration of the devices (closed technology) has limited the development of customized network applications and QoS services.

Software Defined Networking is a new network paradigm that removes the rigidity present on current architectures and improves flexibility and management in networks. SDN decouples the control plane and the data plane in network devices and establish an open communication interface between them. In addition, SDN proposes a centralized control of the network and open APIs to facilitate the development of high level network applications and services. OpenFlow is the first SDN standard that has been widely used in different research projects [9] [10]. OpenFlow is designed based on the actual flow tables located in traditional network devices and opens those up. The controller uses the OpenFlow protocol [11] to configure the flow tables in switches. Figure 1 shows the differences between SDN and traditional architectures.

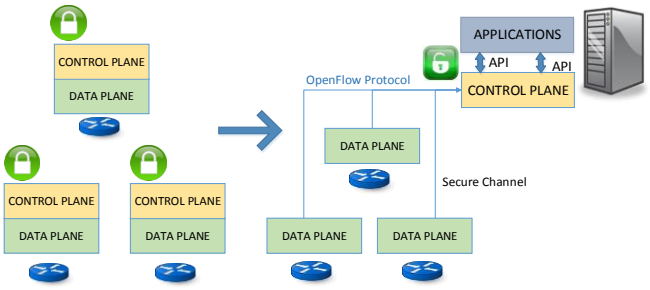


Fig. 1. Comparison between traditional and SDN architectures [10]

Another limitation of the actual infrastructure is the highly amount of network devices, each operating their own private software and highly dependent in proprietary hardware. For this reason, the design and installation of new services usually require the individual software updating or the replacement of hardware. This rigidity increases the installation and operational costs. In this context, the Network Function Virtualization NFV concept has gained in importance in the telecommunications industry.

Network Function Virtualization proposes the transferring of the different network functions (routing, firewall, deep packet inspection DPI, gateway) as virtual software-based applications executed in IT platforms (servers, switches and storage). This new vision of IT services provides a major flexibility and scalability, facilitates the development cycles and reduce costs. Figure 2 describes the differences between NFV and traditional architectures.

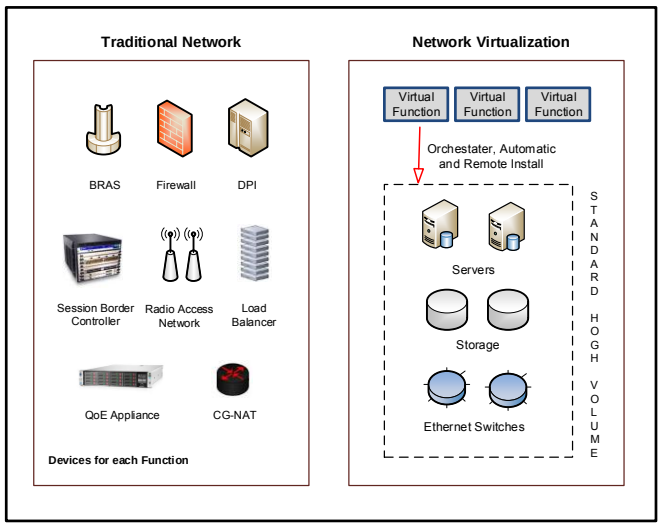


Fig. 2. Comparison between traditional and NFV paradigms

The NFV architecture identifies three principal modules: the Network Function Virtualization Infrastructure (NFVI) that includes all hardware resources, the Virtualized Network Function (VNF) that represents a network functions executed over the NFVI and the NFV Management and Orchestrator (NFV M&O) responsible for coordinate the execution of the different network functions (NF) over the infrastructure [12].

It is important to note that SDN and NFV concepts are different but complementary. Thanks to the SDN separation of data and control planes in network devices, the integration of NFV in virtualized IT environments is feasible. However, the implementation in production networks has several challenges to be addressed [13].

III. Integration of Wireless Networks with SDN/NFV

In the last years, SDN approach has been expanded to mobile networks, giving rise the concept of Software Defined Wireless Networks [5] [14] [15] [16] or Wireless Mesh Software Defined Networks [17] [18]. Similarly, the integration between SDN and mobile technologies (LTE) has gained the attention of industry and research community [19] [20] [21] as well as the close relationship between SDN and NFV. For instance, EmPOWER [22] shows a testbed composed of 30 nodes that facilitates the deployment of SDN/NFV experiments for WIFI networks and it also provides monitoring tools in order to control the energy consumption. In the wireless field, some research intends to apply OpenFlow in order to enhance these types of networks.

doi:10.15849/icit.2015.0038

OpenRoads [5] presents an architecture of three layers (flow, slicing and controller) based on OpenFlow protocol and SNMP in order to innovate in WIMAX and WIFI networks. In the same way as OpenFlow wired networks, the wireless devices (Access Point or Wimax base station) have a flow table which are controlled through the controller. For its part, the slicing layer divides the data traffic through the FlowVisor tool [23] and NOX controller is the brain of the network control. The deployment contains 85 Access Points and two Wimax Base stations (over Stanford campus network) and provides functions such as hard handover, bicasting, Hoolock, among others.

Dely et al. in [18] presents an approach to improve the mobility in Wireless Mesh Networks (WMN). For this purpose, it introduces a mesh router known as a Mesh Access Point (MAP), which forwarding the traffic to the destination through other mesh routers or gateways. It is important to note that MAPs have OpenFlow support. Each node has some physical wireless cards and these in turn are divided into two virtual interfaces, one related with the data plane and the other for control traffic. The data interface is related with OpenFlow datapath. The gateways allow connectivity with the outside and each mesh router has an agent in order to monitoring the links, channel utilization, and others. Moreover the core network has two elements: the Monitoring and Control Server (MCS) and NOX controller. MCS builds a topology database with the information from mesh routers and NOX controls the mesh network. Proofs of concept were conducted over an experimental Wireless Broadband Mesh Network (KAUMesh), which is based on 802.11a/b/g standard. These tests were focused on mobility capacities, when clients move rapidly between different MAPs.

Regarding to cellular networks, the advances are limited and not homogeneous due each research applies different approaches and focuses on diverse elements, the Radio Access network (RAN) and the Evolved Packet Core (EPC).

On one hand, SoftRAN [24] framework uses the SDN concept in order to improve the RAN performance. SoftRAN has a whole view of interference and load of each node and in this way coordinates the allocation of radio resources, especially in dense networks. Each base station sends periodically information to controller and it is saved in a data base, which contains the following elements: an interference map, the flow records and the network operator preferences. SoftRAN was tested with some use cases such as load balancing and utility optimization.

On the other hand, there are some approaches focused on the core part of cellular networks. CellSDN [25] [26] provides an architecture with advances characteristics, such as the slicing of the network resources, better packet classification through deep packet inspection functionalities, scalability via local switch agents and the creation of applications based on the user attributes (network provider, device type).

For its parts, SoftCell [6] enhances the scalability and flexibility in SDN/LTE networks through the analysis of workload and the implementation of fine-grained policies. Softcell also aggregates the traffic based on different aspects such as the base station, mobile devices and the service polices. Each base station is connected with an access switch. This switch has OpenFlow support and is supervised by the controller.

Similarly, MobileFlow [21] takes advantage from SDN and data center concepts to enable and foster the innovation in carrier networks. The main elements of the architecture are MobileFlow Forwarding Engine (MFFE) and MobileFlow Controller (MFC). MMFE has support to mobile network tunnel capacities and allows the integration with legacy EPC equipment. For this reason, the MMFE is considered the data plane. Each MMFE is controlled by the MFC (control plane). The implementation and validation process consists on a prototype based on x86 servers and OpenFlow components.

Furthermore, some projects could be applied to SDN/mobile networks (Wireless and cellular). These projects not only take into account SDN technology but also another key enabler technologies such as NFV, cloud computing, advances virtualization techniques, among others.

For instance, T-NOVA project [7] aims the design and implementation of a framework to allow operators the deployment of virtualized Network Functions (NF) over Network/IT infrastructures. This virtual network appliances are developed in software using SDN/NFV and eliminate the need of acquire, install and maintain specialized hardware. The framework will enable an open API for developers to the design and develop of NF appliances.

UNIFY (Unifying Cloud and Carrier Networks) [8] considers the entire network (home networks to data centers) as a "unified production environment", focusing on telco functions. UNIFY combines the benefits of cloud computing and virtualization in order to build a new architecture that optimizes data traffic flows and allows the dynamic placement of networking, computer and storage components. The consortium creates a model with advanced programmability, new languages, algorithms and management tools to optimize data traffic across networks. UNIFY intends to design a universal hardware node in order to support network functions and traditional data center workloads. The whole architecture allows agility (velocity), simplicity (automation), flexibility (granularity) and programmability of the services, providing and open environment for the deployment of these services and, at the same time, reducing the costs. UNIFY will derive a framework which supports a variety of services such as OpenFlow, Network Function Virtualization, and so on. Besides, this project is focused on three areas. First, infrastructure virtualization, second, flexible service chaining and thirdly, network service chain invocation (programmability interfaces).

CROWD project (Connectivity management for eneRgy Optimised Wireless Dense networks) [27] proposes a novel architecture in order to enhance very dense and heterogeneous wireless networks (Dense Nets). CROWD promotes a paradigm change in this kind of networks through global network cooperation, fine and dynamic network configuration,

doi:10.15849/icit.2015.0038

resources on demand, among others. For this purpose, this project uses SDN and OpenFlow protocol as an enabler concepts to control and manage in an efficient way the resources of Dense Nets. CROWD architecture has two kind of controllers: local and regional. The infrastructure layer consists of base stations (eNBs or Wifi AP) which are configurable via OpenFlow. CROWD provides dynamic controller placement, dynamic backhaul reconfiguration, energy optimization, MAC optimization mechanism and ensures user quality of experience.

CITYFLOW project (OpenFlow City Experiment – Linking Infrastructure and Applications) [28] introduces the use of virtual path slice (VPS) technology at large scale on an OpenFlow network. This project emulates a city with one million inhabitants, with OpenFlow support and taking into account network topologies over xDSL, LTE and Fiber technologies.

Moreover, there are some facilities that allow the experimentation with SDN in wireless environment, such as OFELIA (Open Flow in Europe: Linking Infrastructure and Applications) [29] and the above mentioned deployments, OpenRoads [5] and KAUMesh [18]. OFELIA provides an environment to investigate and validate revolutionary ideas. OFELIA has a set on ten islands over Europe based on OpenFlow technology. Likewise, SmartFIRE [30] develops a large-scale testbed located in South Korea and Europe. The European zone is composed by three different and heterogeneous islands. Two islands belong to the OFELIA project (iMinds and UMU) and the other is part of the OpenLab federation testbed (UTH testbed). For its part, South Korea testbed includes OpenFlow islands in different institutes, for example Electronics and Telecommunications Research Institute (ETRI) and Seoul National University (SNU).

All these advancements are in an early stage but the initial results are promising. Next, we present the current trends and challenges and a possible architecture aligned with the SDN and NFV concepts.

IV. TRENDS AND CHALLENGES

Nowadays, the variety of mobile networks providing different services and applications requires a mobile infrastructure capable of provide high levels of security, performance and QoS. This means that current mobile networks requires a standardized environment, wherein foster the innovation and introduction of new services would be possible in less time and with the lowest investment. This may be achieved through the synergy of NFV and SDN. On one hand, SDN enhances the control and management of network devices through the centralized control. On the other hand, NFV reduces the investment by means of sharing resources not only physical infrastructures but also network functions. This means the reduction in capital (Capex) and operational costs (Opex) that is the main limitation of carrier and service providers. In the context of mobile networks, there are some challenges that a SDN/NFV approach may solve. Next, we describe the actual issues and trends.

Rapid innovation: The combination of SDN and NFV reduces the time to market of new services, through the resource virtualization and centralized control in different locations over an standardized environment. This eliminates the vendor dependence and increases the benefits for stakeholders.

Mobile traffic monitoring and management: SDN allows fine-grained control of the network data traffic and resources. This is especially important for handover, where OpenFlow may facilitate the change between nodes. Additionally, the traffic could be classified and managed based on the kind of flow, aggregation criteria (cell, user equipment, etc), flow rate, occupation of the resources (channels, links, base stations or AP, available bandwidth), among others. For instance, could be possible connect users to multiple networks or defines threshold parameters (bandwidth, location), allowing the easy change between them. Other applications may include the dynamic resources management of wireless backhaul or the capacity aggregation not only with one technology but also combining different technologies.

Energy efficiency: The traffic load is changing constantly according different factors, time, location, special events, among others. On one hand, SDN may enable the optimization of the power consumption based on real time conditions. In this way, the SDN controller could increase or decrease the number of resources allocated. On the other hand, NFV may decrease the number of devices due its flexibility and sharing capabilities.

Scalability and flexibility: Nowadays, the introduction or extension of new services is not easy because current architectures are closed. It process requires a long time or in some case is not performed due the investment is greater than the economy benefits SDN/NFV facilitates the service scalability and allows the reutilization of infrastructures and applications. Moreover, mobile networks are more flexible because SDN/NFV approach is aware of network conditions and changing traffic patterns.

Sharing infrastructure or services: SDN monitoring and VNF virtualization capabilities enable to share infrastructures and network resources. A service provider (SP) could deploy their network functions in the infrastructure of another SP, or use the applications (of another SP) in their own infrastructure. All of these activities are managed by the SDN controller. As a result of this, SDN/NFV introduces new capabilities in billing services. This generates more revenue for stakeholders, the first SP obtains revenue from the service and the second with the infrastructure lease. However, this is an ideal environment to network business; there are some legacy concerns that would be solved or agreed before this scheme can perform.

Inter-Cell Interference: Several APs or base stations in the same location could produce interference each other due the cell overlapping, bad coordination of subcarriers, among others. Consequently, it produces degradation of quality of services (QoS). In this context, SDN enables the easy

doi:10.15849/icit.2015.0038

management of radio resources by means the centralized control and the global view of the network.

Security: The full picture of network events of SDN allows a better control and the detection of anomalous activities. The controller could provide pieces of software that acts like Intrusion Detection System (IDS), firewalls or another security function.

The advances in these concerns are in preliminary state and require the effort and coordination of vendors, researchers and the organism working in these areas, such as Open Networking Foundation (ONF), International Telecommunication Union (ITU) or the European Telecommunications Standards Institute (ETSI), among others [31].

The Wireless and Mobile Working Group (WMWG) aids to promote and extend the ONF approaches in this field, it includes the incorporation of OpenFlow protocol with mobile networks, following the current standards such as 3GPP, IEEE and others. ITU tries to standardize SDN for telecom carriers. For instance, Joint Coordination Activity on SDN (JCA-SDN) coordinates the ideas from different Standard Developing Organizations (SDO) and open sources activities. Other group (SG11) is discussing SDN signaling. For its part, Internet Research Task Force (IRTF) has created the Software-Defined Networking Research Group (SDNRG) and Network Function Virtualization Research Group (NFVRG), which analyze the approaches that can be used in both technologies. Moreover, ETSI-ISG has delivered some initial requirements, service models and use cases for Network Function Virtualization.

We have presented the benefits of SDN/NFV in mobile networks. Based on the premises of both technologies, a whole overview of a possible framework is shown in Figure 3.

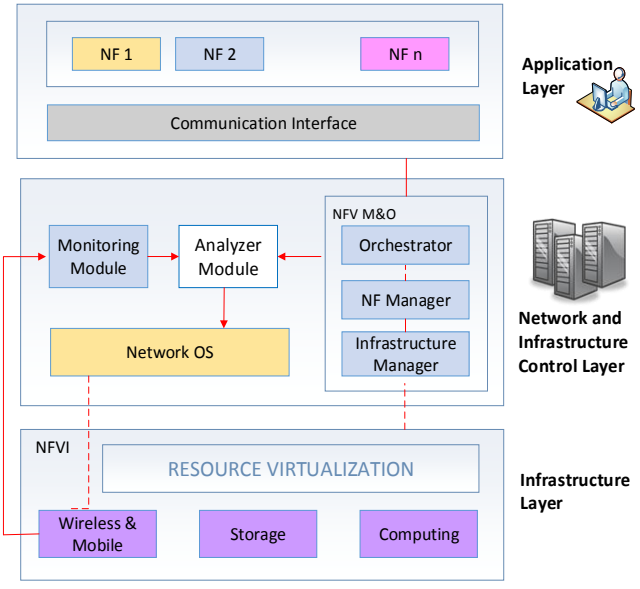


Fig. 3. SDN/NFV Architecture.

This architecture takes into account SDN and NFV technologies. On one hand, the framework presents a layered structure: data, control and application layers, in the same way that SDN architecture. Moreover, it takes advantage of NFV concept to allow the easy implementation and management of network functions, without the need to increase the hardware devices.

In the data layer, we have the current mobile infrastructure of the network operator providing support to a wide range of wireless and cellular technologies like WiFi, LTE, UTMS, GSM, among others. On top of this hardware layer, there is a virtualization layer to enable the virtualization of hardware devices. The resources could be in different locations and data centers and takes into account three components:

- *Networking*: These devices incorporate mobile technologies and OpenFlow protocol.
- *Storage*: This element can include Object storage or block storage (Swift and Cinder OpenStack) or another novel techniques.
- *Computing*: It include high volume servers. It could also use Openstack Nova.

The control layer is in charge of monitoring, analysis, management and orchestration of devices. Consist of four modules: monitoring, analyzer, network OS and NFV M&O.

- *Monitoring Module*: This module is able to provide the complete low-level overview of the managed systems by mean of gathering metrics coming from different network devices.
- *Analyzer Module*: This module could give a deep analysis of the data in order to determine the suited behavior of the network. This module also can infer the recommended behavior of the network. The techniques used in the analysis can include: data mining, learning algorithms, pattern recognition, among others.
- *Network OS*: This module control de basic functions of the control layer. Also, it uses the OpenFlow or similar protocols to send instruction to the Infrastructure Layer elements. Its functionality is similar with an Operating Systems OS in computing.
- *NFV M&O*: This module determines and organizes the actions to be executed in the system, the orchestration, the management of the resources and the control functions.

On the top of the architecture is located the application layer, which consists of two basic modules:

- *Communication Interface*: This module enables an open API to programmers to facilitate the development of new services.
- *Network Functions*: This module presents an scalable structure to create customized network functions or control applications.

This architecture enables users and developers a global view of IT infrastructure. Furthermore, the elements located on

doi:10.15849/icit.2015.0038

SDN/NFV control layer can adapt the network resources depending on the actual situation of the network and dynamically respond to failures or degradation of network performance.

V. CONCLUSION

The integration of digital services distributed over multiple mobile devices (laptop, tablet, cell phone, IoT) sharing high amounts of data (VoIP, streaming, digital images, e-gaming) have been limited by the closed-access and rigidity of actual IT infrastructure. Software Defined Networking and Network Function Virtualization have emerged as a part of the solution for the openness of the infrastructure and enabling to network administrator the dynamically customization of the network behavior.

This work presents a whole overview of the limitations of current IT infrastructure and introduces the novel concepts of SDN and NFV. Similarly, we describe the recent projects based on the integration of SDN/NFV with mobile infrastructure. The current trends and challenges in order to implement these advances in production networks are analyzed. Finally, we present an SDN/NFV architecture that integrates mobile and wireless technologies. It is clear that these paradigms bring new opportunities and create new business models for users, operators and service providers. However, it is fundamental the coordination between research community, industry and service operators in order to implement these advances in production networks.

ACKNOWLEDGMENT

The research leading to these results has been partially funded by the European Union's H2020 Program under the project SELFNET (671672). Ángel Leonardo Valdivieso Caraguay and Lorena Isabel Barona López are supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 and 2013 Scholarship Program. This work was partially supported by the "Programa de Financiación de Grupos de Investigación UCM validados de la Universidad Complutense de Madrid – Banco Santander".

REFERENCES

[1] B. Naudts, M. Kind, F. Westphal, S. Verbrugge, D. Colle, M. Pickavet. "Techno-economic Analysis of Software Defined Networking as Architecture for the Virtualization of a Mobile Network," *in Proceedings of the European Workshop on Software Defined Networking*, EWSDN, Darmstadt, Germany, October 2012, pp. 67-72.

[2] J. Q. Wang, F. Haijing, C. Chang. "Software Defined Networking for Telecom Operators: Architecture and Applications," *in Proceedings of the 8th International Conference on Communications and Networking in China*, CHINACOM, Guilin, China, August 2013, pp. 828-833.

[3] B. Haberland, F. Derakhshan, H. Grob-Lipski, R. Klotsche, W. Rehm, P. Schefczik, M. Soellner. "Radio Base Stations in the Cloud," *in Bell Labs Technical Journal*, vol. 18, no. 1, June 2013, pp 129-152.

[4] J. Costa-Requena. "SDN Integration in LTE Mobile Backhaul Networks," *in Proceedings of the International Conference on Information Networking*, ICOIN, Phuket, Thailand, February 2014, pp. 264-269.

[5] K. K. Yap, M. Kobayashi, R. Sherwood, T. Y. Huang, M. Chan, N. Handigol, N. McKeown. "OpenRoads: Empowering Research in Mobile Networks," *in ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, January 2010, pp. 125-126.

[6] X. Jin, L. E. Li, L. Vanbever, J. Rexford. "Softcell: Scalable and Flexible Cellular Core Network Architecture," *in Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ACM, California, CA, USA, December 2013, pp. 163-174.

[7] TNOVA EU project, http://www.t-nova.eu/.

[8] UNIFY EU project, http://www.fp7-unify.eu/.

[9] A. L. Valdivieso Caraguay, L. I. Barona López, L. J. García Villalba. "Evolution and Challenges of Software Defined Networking," *in Proceedings of the Workshop on Software Defined Networks for Future Networks and Services*, Trento, Italy, November 2013, pp. 47-55.

[10] A. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, L. J. García Villalba. "SDN: Evolution and Opportunities in the Development IoT Applications," *International Journal of Distributed Sensor Networks*, IJDSN, May 2014, pp. 1-10.

[11] N. McKeown, T. Anderson, H. Balakrishnan, et al. "OpenFlow: Enabling Innovation in Campus Networks," *in ACM SIGCOMM Computer Communication Review*, vol. 38, no.2, April 2008, pp. 69-74.

[12] ETSI Industry Specification Group (ISG). "Network Function Virtualization (NFV) White Paper," *in SDN and OpenFlow World Congress*, Frankfurt, Germany, September 2013, pp. 1-16.

[13] H. Hawilo, A. Shami, M. Mirahmadi, R. Asal. "NFV: State of the Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC)," *in IEEE Network*, vol. 28, no. 6, November 2014, pp. 18-26.

[14] S. Costanzo, L. Galluccio, G. Morabito, S. Palazzo. "Software Defined Wireless Networks: Unbridling SDNs," *in Proceedings of the European Workshop on Software Defined Networking*, EWSDN, Darmstadt, Germany, October 2012, pp. 1-6.

[15] C. Chaudet, Y. Haddad. "Wireless Software Defined Networks: Challenges and Opportunities," *in Proceedings of the 2013 IEEE International Conference on Microwaves, Communications, Antennas and Electronics Systems*, COMCAS, Tel Aviv, Israel, October 2013, pp. 1-5.

[16] K. K. Yap, R. Sherwood, M. Kobayashi, T. Y. Huang, M. Chan, N. Handigol, G. Parulkar. "Blueprint for Introducing Innovation into Wireless Mobile Networks," *in Proceedings of the Second SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, ACM, New Delhi, India, September 2010, pp. 25-32.

[17] A. Detti, C. Pisa, S. Salsano, N. Blefari-Melazzi. "Wireless Mesh Software Defined Networks (wmSDN)," *in Proceedings of the 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, WiMob, Lyon, France, October 2013, pp. 89-95.

[18] P. Dely, A. Kassler, N. Bayer. "Openflow for Wireless Mesh Networks," *in Proceedings of the 20th IEEE International Conference on Computer Communications and Networks*, ICCCN, Maui, Hawaii, July 2011, pp. 1-6.

[19] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, E.-D. Schmidt. "A Virtual SDN-Enabled LTE EPC Architecture: A Case Study for S-/P-Gateways Functions," *in Proceedings of the Workshop on Software Defined Networks for Future Networks and Services*, Trento, Italy, November 2013, pp. 8-14.

[20] S. B. H. Said, M. R. Sama, K. Guillouard, L. Suciu, G. Simon, X. Lagrange, J. M. Bonnin. "New Control Plane in 3GPP LTE/EPC Architecture for On-Demand Connectivity Service," *in Proceedings of the Second IEEE International Conference on Cloud Networking*, CloudNet, San Francisco, SF, USA, November 2013, pp. 205-209.

[21] K. Pentikousis, Y. Wang, W. Hu. "Mobileflow: Toward Software-defined Mobile Networks," *in IEEE Communications Magazine*, vol. 51, no. 7, July 2013, pp. 44-53.

[22] R. Riggio, T. Rasheed, F. Granelli. "EmPOWER: A Testbed for Network Function Virtualization Research and Experimentation," *in Proceedings of the Workshop on Software Defined Networks for Future Networks and Services*, Trento, Italy, November 2013, pp. 138-142.

doi:10.15849/icit.2015.0038

[23] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. M. Parulkar. "Can the Production Network be the testbed?," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, OSDI, Vancouver, BC, Canada, vol. 10, October 2010, pp. 365-378.

[24] A. Gudipati, D. Perry, L. E. Li, S. Katti. "SoftRAN: Software Defined Radio Access Network," *in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ACM, Hong Kong, China, August 2013, pp. 25-30.

[25] L. E. Li, Z. M. Mao, J. Rexford. "Toward Software-Defined Cellular Networks," *in Proceedings of the European Workshop on Software Defined Networking*, EWSDN, Darmstadt, Germany, October 2012, pp. 7-12.

[26] L. E. Li, Z. M. Mao, J. Rexford. "CellSDN: Software-Defined Cellular Networks," *in Technical Report*, Princeton University, 2012.

[27] ICT-CROWD EU project, http://www.ict-crowd.eu/.

[28] CITYFLOW project, http://www.onesource.pt/cityflow/site/.

[29] OFELIA EU project, http://www.fp7-ofelia.eu/.

[30] EUKOREA EU project, http://eukorea-fire.eu/pilots/.

[31] F. Schneider, T. Egawa, S. Schaller, S. I. Hayano, M. Schöller, F. Zdarsky. "Standardizations of SDN and Its Practical Implementation," *in NEC Technical Journal*, vol. 8, no. 2, April 2014, pp. 16-20.

An Optimization Framework for Monitoring of SDN/OpenFlow Networks

Ángel Leonardo Valdivieso Caraguay, Jesús Antonio Puente Fernández, Luis Javier García Villalba

Abstract

The centralized control of the network and the separation of data and control planes proposed by Software Defined Networking SDN have changed the rigid, static, and complex nature of the networks. The decisions taken by the control plane depends on the accuracy of the monitored information on network performance and detection of network events (link failure, delay, loss, network overhead). However, the monitoring information is typically provided by external network monitoring solutions which require the installation of specialized (and costly) equipment.

This work proposes an efficient SDN monitoring framework using the OpenFlow protocol. This framework uses profiling to provide different monitoring levels based on the requirements of the "network programmer". Moreover, the pluggable architecture enables the creation, updating and customization of high level metrics as well as the orchestrator balancing the monitoring tasks and offering an adaptive method of polling information requests based on the load of the controller and the size of the network. The implementation of network performance metrics (data rate, loss rate and delay) and the results of experiments using video streaming traffic demonstrate the effectiveness of the framework.

Index Terms

Framework, Quality of Service, Monitoring, Multimedia, OpenFlow, SDN.

I. Introduction

The Software Defined Networking SDN architecture opens up the possibility to customize the network behavior depending on the actual network conditions. The complexity, rigidity and closed union between hardware and software of the network devices are replaced by an open API between data and control planes. Moreover, the individual treatment of the packet in a node is also replaced by a centralized control of the network. Through it, the network programmer, is able to design and literally program network applications and services. OpenFlow [1] is the main SDN protocol used to communicate between the data plane (switches) and the control plane (controller). This protocol opens the remote control of the flowtables present in the network devices and establishes a secure channel to the controller. This protocol is widely used in different research projects such as home networking, data centers, mobility, and security, amongst others [2], [3].

The good operation of the network also depends on the controller capacity to monitor the network performance and correctly detect critical network events (link failure, delay, loss). The accuracy of this information depends on the measurement technology applied to the network infrastructure. Unfortunately, the implementation of this solution requires the installation of new hardware and increases the operational costs. In this context, a feasible solution is the use of the own OpenFlow messages to estimate the network performance. In this work, we analyze the first initiatives and strategies used to implement monitoring solutions using OpenFlow.

The authors are with the Group of Analysis, Security and Systems (GASS), http://gass.ucm.es/en, Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases 9, Ciudad Universitaria, 28040 Madrid, Spain (e-mail: {angevald, jesusantoniopuente, javierv}@ucm.es).

However, one of the main challenges of the OpenFlow based monitoring tasks is the workload of data and control planes. The continuous request of information to every switch of a topology can provide accuracy in the results, but at the same time, can cause high CPU utilization and negatively affect the performance of other tasks. Similarly, the switches (data plane) use more hardware resources and energy to process the controller request. For its part, not all network applications require the same metrics to make decisions. Some services are delay sensitive (video streaming), while others are not (e-mail). Such diversity creates the need to organize the metrics requested to switches in function of the needs of users.

In this context, this work proposes a network monitoring framework that intends to reduce the CPU load and hardware resources whilst maintaining high levels of accuracy. We propose an orchestrator module with an adaptive method of polling information requests based on the controller load and network size. This framework also creates profiles to distribute the metrics in function of the particular user requirements. Furthermore, the architecture provides a pluggable design to facilitate the creation and continuous improvement of the monitoring algorithms. This work also presents algorithms focused on monitoring the data rate, loss rate and delay using passive and active methods. The feasibility of the framework is tested using an implementation in an OpenFlow controller, a network emulator and video server in order to monitor network behavior with video streaming information. Finally, we present the results and describe the conclusions and future work.

The rest of this paper is organized as follows: The related work of monitoring tools with SDN is presented in Section II. Section III describes the SDN/OpenFlow monitoring framework. The description of implementation process is explained in Section IV. Section V describes an application scenario and presents the tests and results. Finally, Section VI presents the conclusions and future challenges.

II. Related Work

The network monitoring solutions have been extensively studied in traditional IP networks. Depending on the strategy used to measure the values, the solutions can be classified as active or passive. In active methods, probe packets are included together with the normal traffic and sent through the network. Then, these probe packets are collected and analyzed to estimate values such as delay, end to end connection status, or round-trip time (e.g. ICMP packets). For its part, passive methods do not add additional packets to the network. The monitor observes the packets without influencing the network performance. This task is performed by agents in devices that observe the traffic (e.g. Number of packets transported by a port) and send the information using a monitoring protocol. Protocols such as SNMP [4] or NETCONF [5] are extensible used to transmit the information from agents that continuously reads the actual state of the network. Also, monitoring tools such as NetFlow [6], sFlow [7] or jFlow [8] apply statistical sampling to estimate flow based measurements. It is clear that the information obtained corresponds to network-layer measurements due the network having no access to user's devices and the corresponding application level metrics.

Furthermore, the OpenFlow protocol enables switches to send information about the state of the switch through OpenFlow messages. This information can be used to estimate the actual situation of the network. For instance, the PayLess project [9] provides a RESTful API to send the flow statistics in different aggregation levels. This work proposes an adaptive algorithm to request statistics and reduce overhead. The link utilization is measured with the information of flow removed and statistic request messages. For its part, our proposal includes encapsulation; profiling as well as the possibility of include different monitoring metrics depending of the users requirements. Furthermore, our proposal presents the implementation of data rate, loss rate and delay measurements.

The MonSamp [10] architecture proposes to send a copy of the traffic to a monitoring agent (collector & analyzer). This agent continuously reads the number of flows in the switch ports and sends the information to the flow sampling algorithm located in the controller. Depending on the link congestion and the capacity of the monitor, the algorithm increases or decreases the number of flow rules used for monitoring purposes using Pyretic (northbound API). Our framework does not duplicate the traffic or add an external agent to

monitor the network behavior. Instead, our proposal uses active and passive strategies using the proper OpenFlow messages to estimate the situation of the network. Similarly, our algorithm balances the load of monitoring tasks in the controller based on the size of the network and CPU load.

OpenNetMon [11] monitors the per flow statistics to obtain end-to-end performance. This work implements measures of throughput, delay and packet loss by polling edge or each path's last switch. The frequency of the request depends on the variation of the measurements with respect to previous values. Furthermore, OpenTM [12] analyzes different querying strategies that can be used to reduce the overhead in OpenFlow switches. The experiments show that the non-uniform distribution querying strategy provides reasonable accuracy. This algorithm randomly selects two switches in the flow path and query information to the closest switch to the destination. Meanwhile, our framework organizes the different algorithms in a pluggable logic as well as the orchestrator balances the load of the data plane based on the size of the network and controller capacity. Furthermore, our architecture proposes the high level analysis of monitoring metrics in order to prevent DDoS attacks, security issues or anomalous behavior.

The solution presented in [13] uses a similar strategy of traceroute. In other words, it uses beacons to send probe packets in order to measure the network performance. The controller installs additional flows in the switches to send packets to other nodes or return to the beacon. Then, the beacon analyzes the returned packets and estimates packet delay or loss rate. In our proposal, the controller uses the information of the OpenFlow messages to estimate different monitoring metrics (passive method). Furthermore, the controller can also act as a software based beacon (active method) to create probe packets and use OpenFlow messages to send probe packets to the switches. Furthermore, the Orchestrator module balances the load of active and passive methods on controller.

III. Framework for Optimized Monitoring

The use of a monitoring framework within the own SDN architecture has several advantages. Firstly, the implementation is independent of the hardware or specific vendor. Once the equipment is compatible with a specific southbound API (e.g. OpenFlow), the complete network can be monitored by the controller. This affirmation does not exclude the use of other network monitoring solutions. Additionally, the rapid response of the controller to network events is fundamental to guarantee an efficient network performance. With more thorough monitoring of the controller, more accurate and timely decisions can be made.

However, the use of the controller to execute monitoring tasks presents several risks that must be considered. For instance, it is necessary to minimize the CPU load in order to avoid interferences in other processes of the controller. In other words, the constant monitoring of all variables in all switches is highly inefficient. It also generates an exponential increment of message requests increasing the load in the data plane (switch hardware). Furthermore, the information required by the controller depends on the network application. For example, real time streaming applications are highly sensitive to throughput and delay variations, while for e-mail services, these measures are basically irrelevant. Moreover, it is also desirable that the algorithms used to monitor these values can be easily added, modified or updated as well as a clear presentation of the results.

Taking into account these aspects, we propose an optimized monitoring framework illustrated in Figure 1. The proposed framework consists of different functional boxes within the control and application layer of the SDN architecture. The infrastructure layer includes the switches, routers and other elements responsible for the forwarding tasks. The control layer makes decisions about the network behavior and sends these instructions to the infrastructure layer through a southbound API (OpenFlow). The control layer offers functionalities to the application layer thanks to a northbound API (Rest API), where high level policies and applications are implemented.

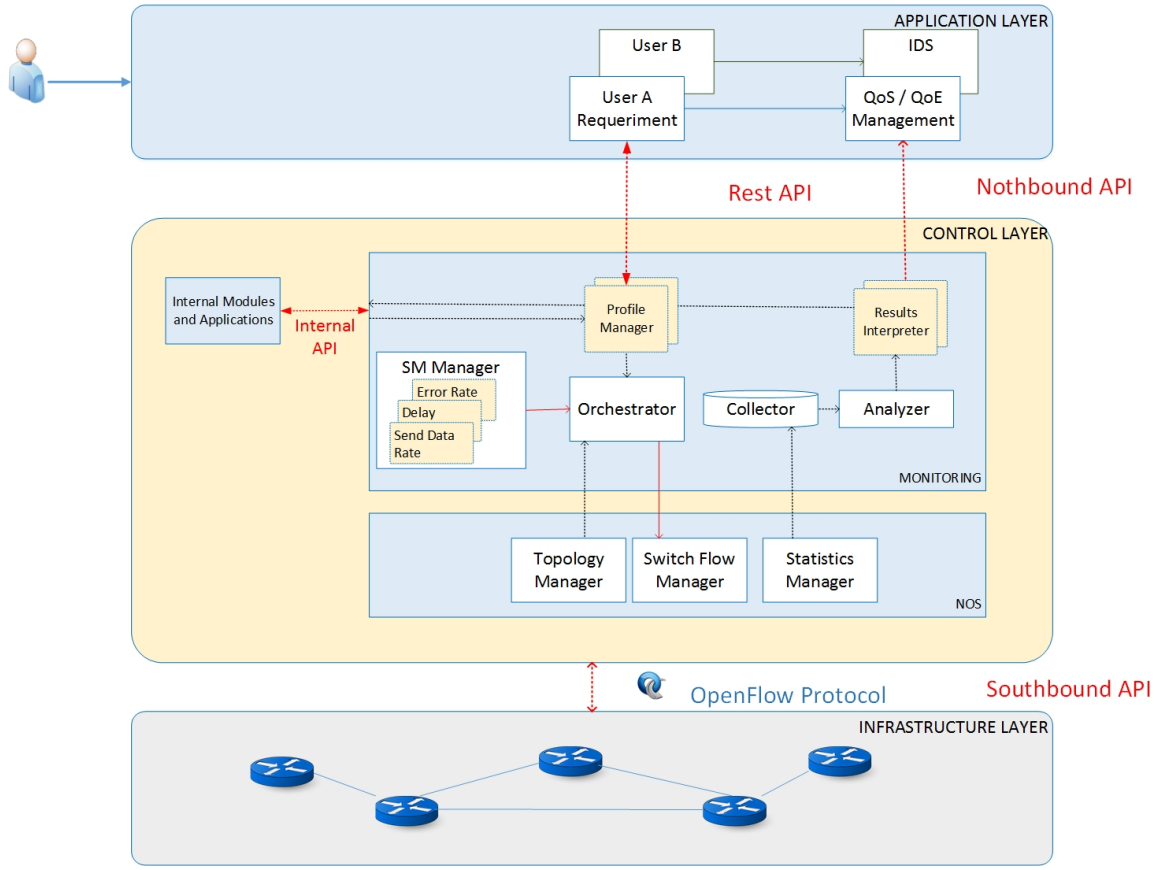


Fig. 1: Monitoring Framework

The functional components of the framework are explained in detail below:

- **User's requirement.** The application that needs information about the network creates a new monitor profile. In this profile, the user specifies the user ID, type of metrics and the level of accuracy. This information is sent to the control layer through a northbound API (Rest API). This ensures that the application will receive updated information of the network performance based on the required metrics. The user can change the configuration of the profile without affecting other modules or changing the network behavior.
- **Profile Manager.** Receives the different requirements of the application layer and creates a monitoring profile for each user. Additionally, this manager also registers the requests of modules located within the control plane (internal modules) through an internal API. The Profile Manager also updates the Results Interpreter module in order to synchronize and provide accuracy in the results sent to the users.
- **Topology Manager.** This module uses the LLDP (Link Layer Discovery Protocol) and analyzes the OpenFlow messages to identify the network devices, their capabilities and the links presented in the infrastructure layer. The topology is organized as a graph $G(N, A)$, where N represent the switches and A the links present in the infrastructure. The information of the topology is sent to other modules of the control layer.
- **SM Manager.** The Store Metrics Manager consists of a container that registers and stores the available monitoring algorithms. These algorithms are pluggable and can be easily added, modified and updated. The SM Manager can include own or third parties modules. The modules use OpenFlow as southbound and can include external APIs to receive information of other external monitoring tools (NetFlow, SFlow).

- **Orchestrator.** This module organizes and schedules the different monitoring modules in function of the requirement of the profile manager. For this purpose, it takes into account the information provided by the topology manager and balances the load of requests in the infrastructure layer. This module also attempts to reduce duplication of request and avoids data redundancy.
- **Switch Flow Manager.** Receives the instructions of the monitoring algorithms organized by the Orchestrator and sends the OpenFlow messages to request information to the switches. Furthermore, this module uses the topology manager information and modifies the flow tables in case of establishing a route for probe packets.
- **Statistics Manager.** Receives the information provided by the OpenFlow messages and recovers the statistical information of these messages. This information is sent to the Collector module.
- **Collector.** This module filters and organizes the information provided by the statistics manager and stores the relevant information. This module can use compression, data mining and search algorithms to reduce the space required to store the data.
- **Analyzer.** Reads the results from the Collector and carries out event processing, data correlation, learning algorithms to provide high level metrics. The high level metrics are focused not only to individual devices (switch) but also in a global view or complex distributed grouping of devices (network domain). Similarly, the analyzer can infer trending and predicted values about the future network behavior.
- **Results Interpreter.** This module receives the results of the analyzer module and infers HoN (Health of Network) events such as actual or future ACL-violations, IDS/Firewall alerts, DDoS attack, intrusion detection or anomalous behavior. Also, it decides if there has been a traffic increase/decrease, and can ask for a bigger bandwidth if necessary.
- **Applications.** The different network applications receive only the relevant high level monitoring information. The users can change the monitor profile in accordance with their needs and dynamically modify the requests on switches. Furthermore, the user is able to apply reactive and proactive actions providing dynamic responses of different network events.

IV. Implementation

In this section, we explain the process used to implement the framework. Since this work is concentrated in the engine applied to reduce the load of the control plane (controller) and data plane (switches), we describe the aspects related with sampling and the distribution of switch requests.

- **Network Operating System (NOS).** The function of the NOS is similar to the concept of Operating System O.S. in computing. That is, the NOS provides a high-level abstraction of information, resources and hardware. It facilitates the creation of applications independently of the hardware design using a high level programming languages. Moreover, the NOS receives and sends the OpenFlow messages and provides basic functions to the other modules. Examples of available NOS include NOX [14], Maestro [15], Floodlight [16], among others. The framework is implemented using the Floodlight controller. Floodlight is based on Java, available with Apache License and extensively supported by a developer's network. It enables a modular programming together with other advantages of Java environment (platform-independent, timers, threads, libraries). Moreover, Floodlight implements some basic modules compatible with OpenFlow protocol version 1.0 such as topology manager, device manager or statistics manager. These modules are available through a Java API to other internal control plane modules as well as a RestAPI to remote application modules.
- **Topology Abstraction.** The switches and links present in the network infrastructure is represented as a graph $G(N, A)$, where N represents the connected devices and A the arcs between them (links) [17]. In this context, the term $arc(i, j) \in A$ symbolizes a link between nodes i and j. Each link present in the topology can be associated with a weight value or link cost. For instance, the term c_{ij} represents the cost associated with the link $arc(i, j)$. It is also assumed that the network is directed, there is no negative cycles in the topology and the switches are connected with a single control point (one controller).

- **Orchestrator.** These elements are responsible for organizing the algorithms used to establish the network behavior. The algorithms are ordered in modules in such a way that they can be easily added, modified or removed without interfering with each other. The SM Manager registers and organizes the modules used to monitor the network, while the Orchestrator authorizes the execution of the SM modules and controls the load of requests on switches. The optimization of the Orchestrator as well as the algorithms of the SM Manager are open challenges and can be constantly improved. Moreover, it is clear that the monitoring tasks can affect the whole performance of the network, so it is required that the orchestrator can control the load of monitoring tasks. In this implementation, the orchestrator will control the load regulating the pulling period of the requests in function of the network size and the controller capacity. The logic of this technique is explained in the Algorithm 1. The user will assign a minimal period monitoring time t_{min} and the load capacity of the controller α. In case of $\alpha = 1$ the controller is able to support high level of load and with α close to 0 the controller capacity is highly limited. For its part, the network size is estimated in function of the number of devices (N) and links (A). All these variables are taken into account to balance the pulling period (t_{orc}).

Algorithm 1: Orchestrator function

Input: network graph $G(N, A)$
minimal period monitoring time t_{min}
controller capacity $0 < \alpha \leq 1$
Result: optimized monitoring time t_{orc}

```
1 procedure OrchFunction
2     calculate the optimized time in function of the network;
3     if α = 1 then
4         t_orc=t_min;
      else if 0 < α ≤ 1 then
5         if A ≤ N then
6             t_max=N*t_min;
          else
7             t_max = (N + A/N).t_min
8         t_orc = (1 − α).t_max
9 end procedure
```

- **SM Manager.** The SM Manager can include own or third parties modules. As detailed in the Related Works section, there are notable OpenFlow -based monitoring algorithms [9], [11], [18] that have been taken into account as a baseline. The present work presents 3 Algorithms allowing us to obtain the metrics of send data rate (dr_{ij}), packet loss rate (lr_{ij}) and delay (d_{ij}). These proposals have been integrated as modules of SM as follows. The send data rate is explained in the *SendDataRate* procedure in Algorithm 2. *SendDataRate* uses the passive method and sends port request to switches and reads the responses from them. The switch request is sent through the controller to switch message OFPT_STATS_REQUEST. The controller sends this message in an optimized period monitoring time of t_{orc}. The switch responds with an OFPT_STATS_REPLY message. The controller uses the function ofp_port_stats to receive the response, identify the counters with the corresponding switch (src node) and port (src port) in the topology and saves this information s_i^k. Then, the send data rate $dr_{i,j}$ for src node - src port is the difference of the sent bytes counter ($s_i^k - s_i^{k-1}$) in the time period t_{orc}.

Algorithm 2: Send Data Rate (SM Manager)

Input: network graph $G(N, A)$
optimized monitoring time t_{orc}
Result: send data rate dr_{ij} for each $arc(i, j) \in A$

1 **procedure** SENDDATARATE
2 Start timer k with period t_{orc};
3 **foreach** *period* $k = 0, 1, 2, 3, \ldots \in t_{orc}$ **do**
4 **foreach** $arc(i, j)$ **do**
5 Read the sent bytes s_i of the source link port with s_i^k=tx_bytes in $ofp\_port\_stats(node,port(i))$;
6 **if** $k > 0$ **then**
7 Calculate the data rate of the link $dr_{ij}=\frac{s_i^k - s_i^{k-1}}{t_{orc}}$;
8 **end procedure**

Similarly, the packet loss rate metric is explained in the *PacketLossRate* procedure (Algorithm 3). *PacketLossRate* is a passive method that also takes the information from the OFPT_STATS_REQUEST message. That is, the controller does not need to send additional requests due to it using the same information saved to calculate the send data rate. For its part, the implementation uses the Topology Manager to discover the corresponding links present in the infrastructure. Each link $arc(i, j)$ is represented with the key (source node - source port - destination node - destination port). With this information, the *PacketLossRate* calculates the difference lr_{ij} of sent bytes in source node - source port s_i^k with the corresponding received bytes from the destination node - destination port r_j^k in the monitoring time period t_{orc}.

Algorithm 3: Packet Loss Rate (SM Manager)

Input: network graph $G(N, A)$
optimized monitoring time t_{orc}
Result: packet loss rate lr_{ij} for each $arc(i, j) \in A$

1 **procedure** PACKETLOSSRATE
2 Start timer k with period t_{orc};
3 **foreach** *period* $k = 0, 1, 2, 3, \ldots \in t_{orc}$ **do**
4 **foreach** $arc(i, j)$ **do**
5 Read the sent bytes s_i of the source link port with s_i^k=tx_bytes in $ofp\_port\_stats(srcnode,srcport(i))$;
6 Read the received bytes r_j of the destination link port with r_j^k=rx_bytes in $ofp\_port\_stats(dstnode,dstport(j))$;
7 **if** $k > 0$ *(each period)* **then**
8 Calculate the packet loss rate of the link with $lr_{ij}=\frac{(s_i^k - s_j^{k-1}) - (r_i^k - r_j^{k-1})}{t_{orc}}$;
9 **end procedure**

In addition, the *Delay* module uses an active method to calculate the time delay. This procedure is explained in the Algorithm 4. At first, the controller takes a timestamp value t_{st}, encapsulates this value in the load of a packet probe Pp, and then sends this information through the link in the source switch i. The controller sends instructions to the switches to identify this probe packet and sends it back to the controller. In this implementation, we use an experimental network protocol number (253) as the key to identify Pp packets. When the packet arrives in the other extreme of the link (destination node j), the switch identifies, encapsulates and sends this packet back to the controller pkt_{in}. The controller identifies the packet, recovers the initial time stamp t_{st} and then compares this value with the actual timestamp t_{ctr}. The delay value is estimated as the difference between the values t_{ctr} and t_{st} for each link $arc(i,j)$.

Algorithm 4: Delay (SM Manager)

Input: network graph $G(N,A)$
optimized monitoring time t_{orc}
Result: delay dl_{ij} for each $arc(i,j) \in A$

```
1  procedure DELAY
2      procedure SENTPROBEPACKET
3          Start timer k with period t;
4          foreach period k = 0, 1, 2, 3, ... ∈ t do
5              foreach arc(i,j) do
6                  Read the actual timerstamp t_st of the controller
7                  t_st = Date.getTime()
8                  Encapsulate t_st within a Probe packet Pp
9                  Pp_{i→j} = newpacketOutMessage()
10                 Pp_{i→j}.setProtocol(253)
11                 Pp_{i→j}.setData(t_st)
12                 SentthePrObepackettothesourceswitch
13                 switch(i).write(Pp_{i→j})
14     end procedure
15     procedure RECEIVEPROBEPACKET
16         Verify that the incoming packet message is part of a probe packet
17             if packetInMessage pkt_in is a Probe packet (Pp_{i→j}) then
18             Read source, destination and timestamp
19             t_st = pkt_in.getData()
20             i = pkt_in.getSource()
21             j = pkt_in.getDestination()
22             Read the actual timestamp of the controller
23             t_ctr = Date.getTime()
24              Calculate the delay value of the linkarc(i,j)
25             d_ij = t_ctr − t_st
26     end procedure
27 end procedure
```

V. Application Scenario

The feasibility of the framework is tested using the topology described in Figure 2. This topology is emulated using python scripts in the Mininet emulator tool 2.1.0 [19]. Mininet enables the creation of custom topologies of OF-switches and links within a simple laptop (Intel core i5 2.4 Ghz, 8GB DDR3 RAM, OS X 10.10). The tests are executed in a VM Linux Ubuntu 13.04 (64 bits, 4 Gb RAM, 2 CPUs, 8 GB HD). The topology is composed by 3 switches and 3 host connected in a linear topology. The links L1 (s1-s2) and L2 (s2-s3) are configured with values of maximal data rate, loss percentage and delay.

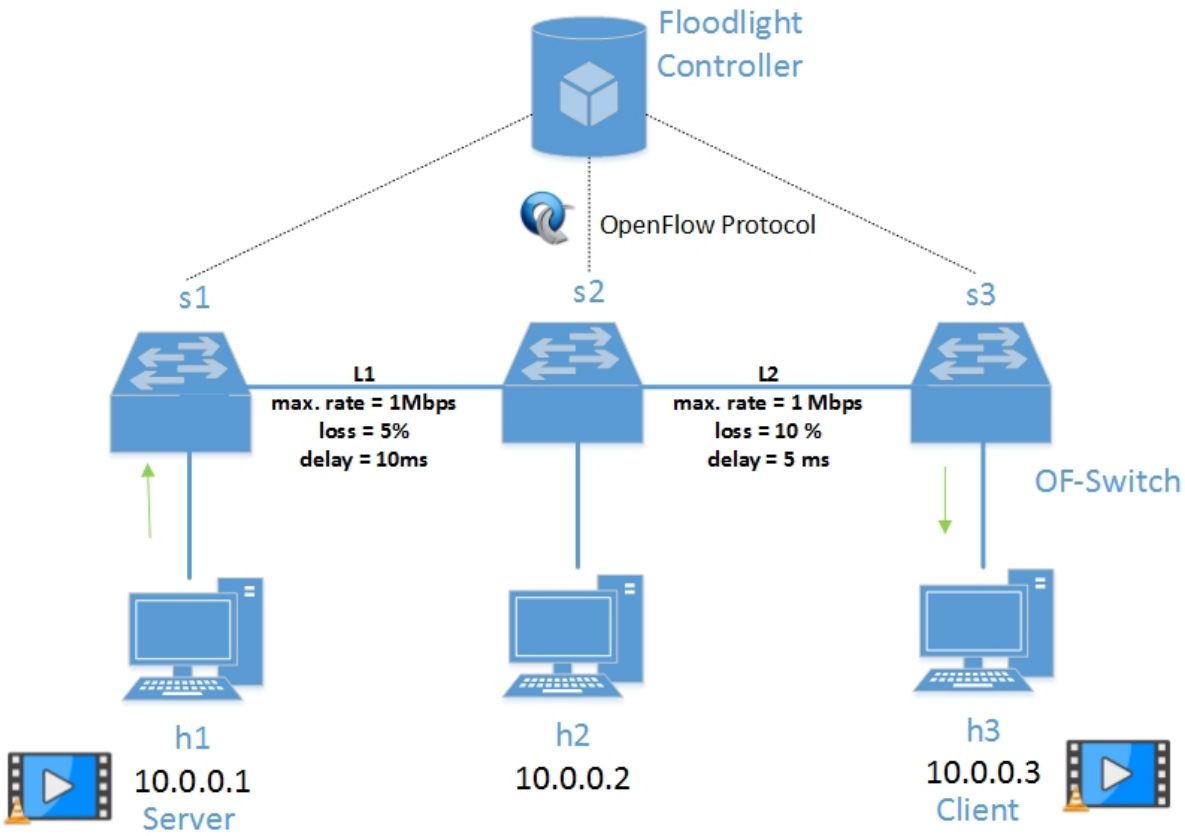


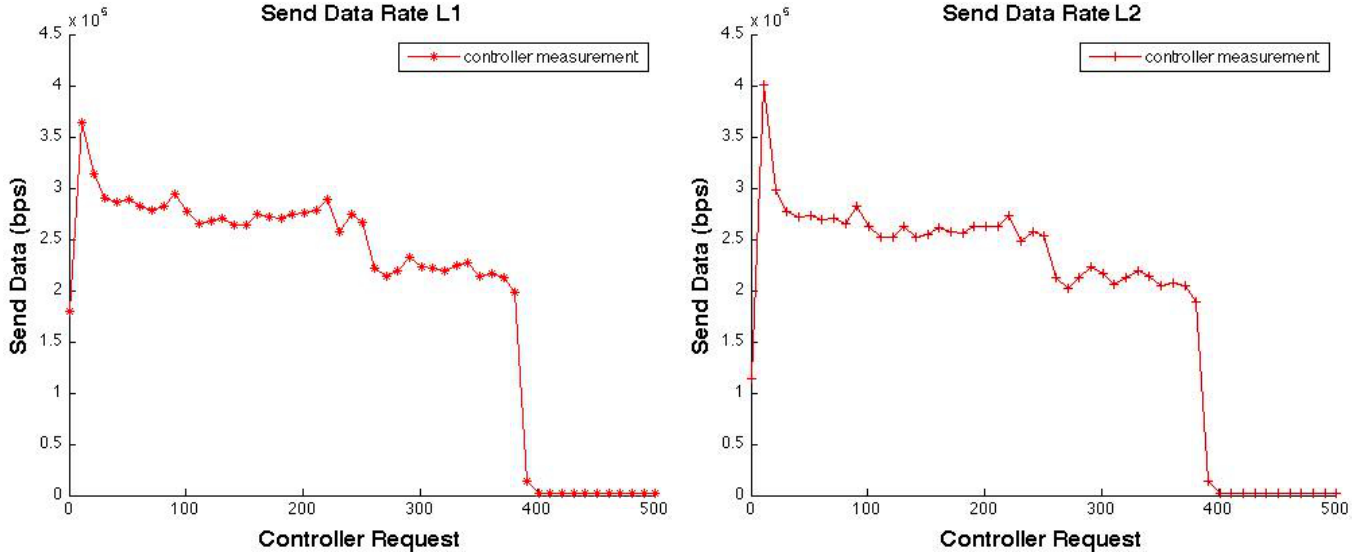
Fig. 2: Test Topology

The host h1 uses VLC video server to send a video file “highway_cif” [20] (MPEG $\frac{1}{2}$, 2000 frames, 80 s, 2.97 MB) to the client h3 using RTP/UDP streaming protocol in order to model traffic. The Floodlight controller executes a learning switch module to establish the path between source and client. Furthermore, the monitoring module is configured with a monitoring time of 200 ms. and a controller capacity $\alpha = 1$. The experiments are executed in the worst-case measurement scenario, that is, the error rate and delay are introduced simultaneously. The representation of the monitoring information provided by the RestAPI northbound is described in the Figure 3. The RestAPI interface provided by the framework enables the rapid development of high level network applications.

{"test": {"1":[{"src-switch":2 , "src-port":2 , "dst-switch":1 , "dst-port":2 , "lk-dataRate":1680 , "lk-errorRate":0 , "lk-delay":14} , {"src-switch":1 , "src-port":2 , "dst-switch":2 , "dst-port":2 , "lk-dataRate":275680 , "lk-errorRate":54800 , "lk-delay":14}], "2":[{"src-switch":3 , "src-port":2 , "dst-switch":2 , "dst-port":3 , "lk-dataRate":1680 , "lk-errorRate":0 , "lk-delay":15} , {"src-switch":2 , "src-port":3 , "dst-switch":3 , "dst-port":2 , "lk-dataRate":220880 , "lk-errorRate":44760 , "lk-delay":16}],"3":[]}}

Fig. 3: Framework RestAPI

The test is repeated 20 times and the average values of data rate, loss rate and delay estimated by controller are evaluated. The results of the experiments are depicted in Figure 4,5,6. In order to reduce distortions, we display the trend line with an average of 10 data. The Figure 4a describes the send data rate metric (calculated by the *SendDataRate* procedure) in bps of L1 (s1-s2) and the Figure 4b the corresponding metric of L2 (s2-s3). As expected, the measured send data rate detects an increase of traffic in the links caused by the video transmission between server and client. Once the transmission is finished (about 400 controller requests), the controller measures a minimal send data rate in both links.



(a) Send data rate for L1 (s1-s2)

(b) Send data rate for L2 (s2-s3)

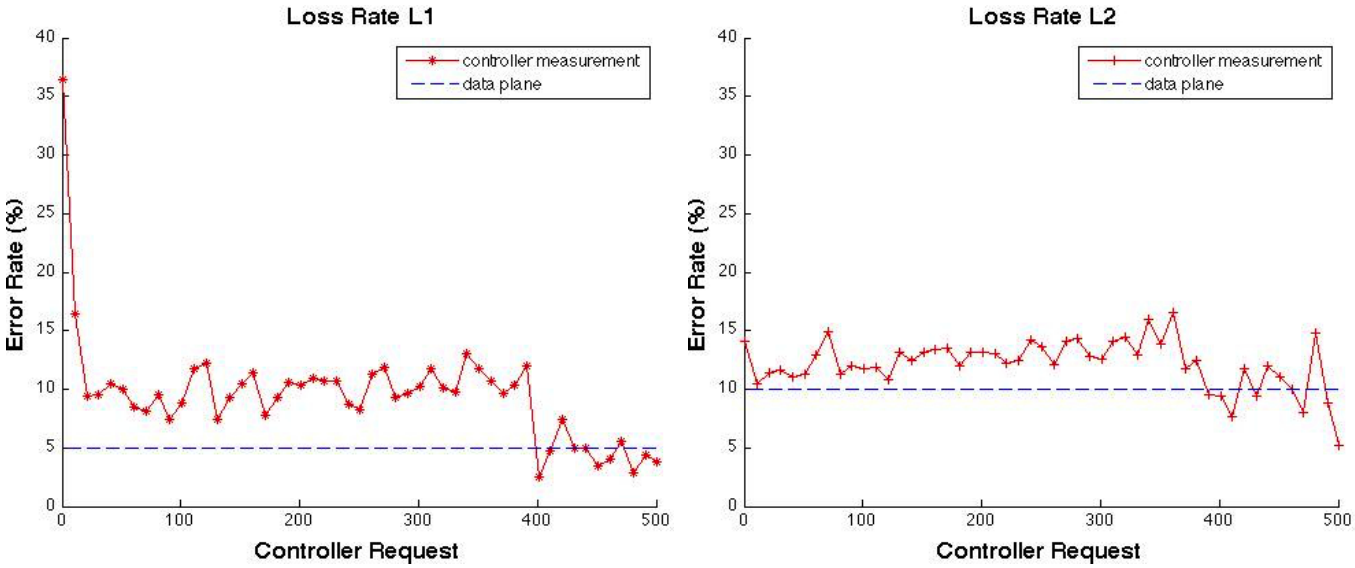
Fig. 4: Send data rate measurement for L1 and L2

Furthermore, the loss rate of the links L1 and L2 are depicted in Figure 5. The red line (solid) represents the loss rate percentage measured by the controller, while the blue line (dotted line) shows the *netem* loss percentage configured in mininet or data plane (L1=5%, L2=10%). In this case, the controller is also capable of detecting the loss of information between links. However, the transmission of video streaming information increases the variation between the data plane and the value estimated by the controller.

The controller delay estimation for L1 and L2 is shown in Figure 6. The blue line (dotted line) shows the data plane delay of L1 (10 ms.) and L2 (5 ms.) and the red line (solid) displays the value measured by controller. In both cases, it can be seen that the measured delay is slightly higher compared with the data plane. This effect is caused by the extra latency between the switch and controller. In other words, for instance, the probe packets in L1 is moving from controller to switch s1, then from s1 to s2 and the switch s2 send the packet back to the controller. This variation can be minimized calculating the delay present between controller and switch using the same active procedure (send and receive a probe packet from the same node) [11]. However, this proposal will increase not only the traffic in switch-controller data path, but also the switch and controller processing tasks.

The Table I presents the nominal data plane of loss rate and delay measurements together with the mean and standard deviation of the results of the experiments. It is clear that, although those values present variations between data and control measurements, the controller can detect problems and quickly send alerts to applications in order to identify causes and take decisions to reduce negative impacts in the network behavior.

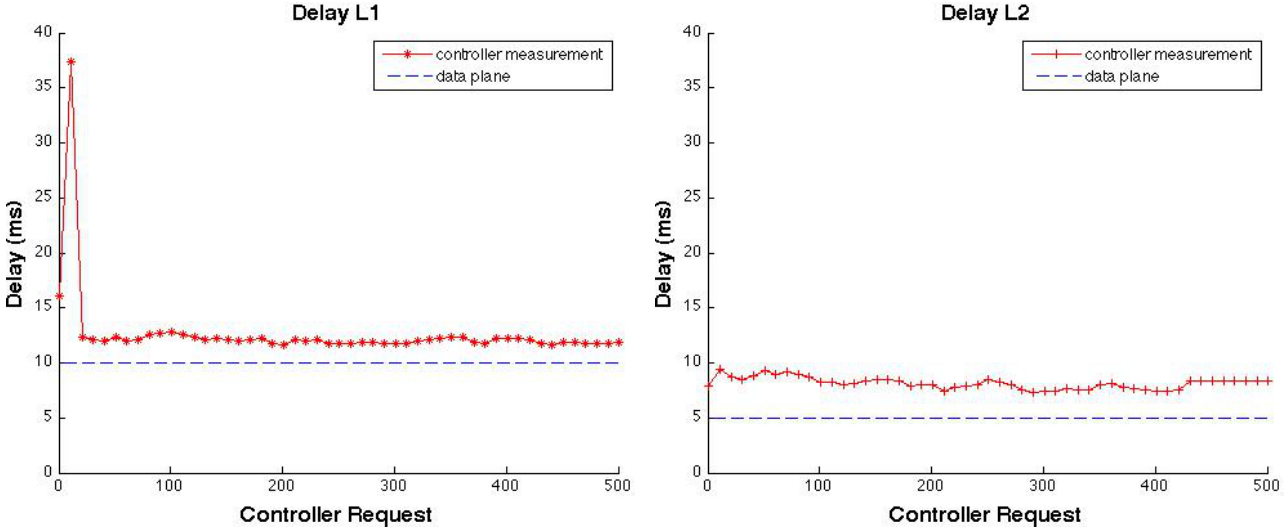
The performance of the framework is also verified in order to analyze the capacity of the controller in different size topologies. The controller capacity is directly linked to the equipment, programming language (Python, Java, C), work load, topology and forwarding complexity [21]. In the work presented in [2], we analyze the performance of different Network Operating Systems (NOS).



(a) Loss data rate for L1 (s1-s2)

(b) Loss data rate for L2 (s2-s3)

Fig. 5: Loss data rate measurement for L1 and L2



(a) Delay for L1 (s1-s2)

(b) Delay for L2 (s2-s3)

Fig. 6: Delay measurement for L1 and L2

	Loss Rate (%)			Delay (ms)		
	data plane	mean	std. dev.	data plane	mean	std. dev.
L1 (s1-s2)	5	9.323	5.716	10	12.626	5.474
L2 (s2-s3)	10	12.623	6.158	5	8.147	0.674

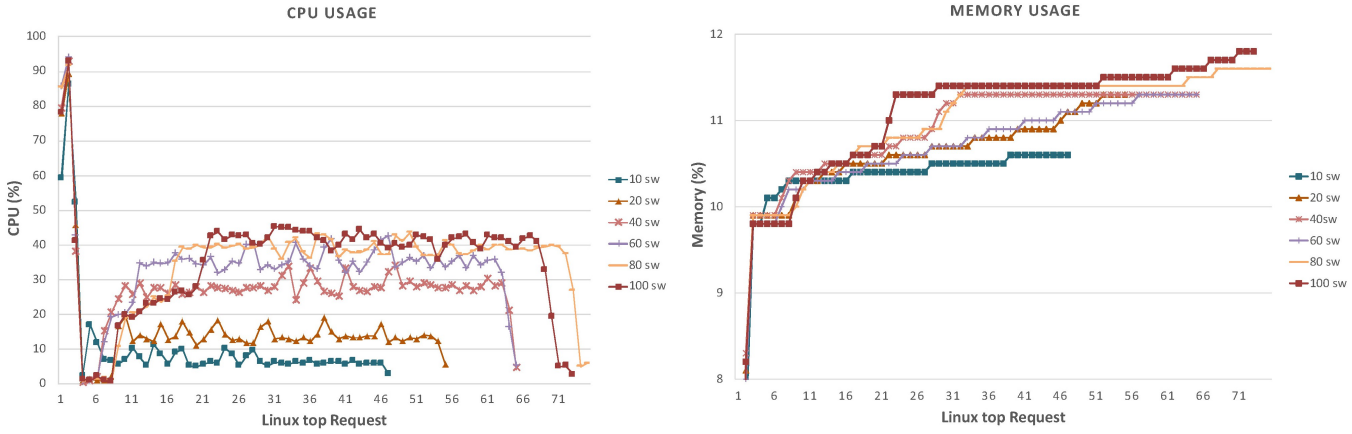
TABLE I: Summary evaluation of the experiment

In the present experiment scenario, the Table II and Figure 7 show the CPU and Memory usage of the controller in different topology sizes, from 10 to 100 switches in a linear topology with an increase of 20 switches between each simulation. These values are measured through system monitoring request using the linux top service. The top requests are sent during the video streaming simulation between the hosts connected in the first and last switch respectively. The CPU usage shows a peak at the beginning

of the simulation as a result of the controller setting up as well as the mutual interference between the processes (Mininet, Floodlight, VLC server and client) running in the same VM. Similarly, as expected, the increase of the network size has an incremental impact in the CPU usage and duration of the streaming simulation. The maximum value measured of CPU usage is 44.4 % for a topology of 100 sw. For its part, the memory usage receive a slight increase with higher topologies (1 %). However, these memory usage percentages are stable during the session. Therefore, one may conclude that the framework is suitable for small-medium scale topologies and in case of large topologies, the framework can be optimized with multi-controller platforms.

		CPU Usage (%)						Memory Usage (%)					
N	Request #	10 sw	20 sw	40 sw	60 sw	80 sw	100 sw	10 sw	20 sw	40 sw	60 sw	80 sw	100 sw
1	1	59.3	77.9	79.5	85.6	85.4	78.1	1.9	1.8	1.9	1.9	1.9	1.9
2	8	6.7	2	20.6	19.3	1	0.7	10.3	9.9	10.3	10.2	9.9	9.8
3	15	8.7	17.3	27.6	34.6	23.6	24.6	10.3	10.4	10.5	10.4	10.5	10.5
4	22	6.3	15.6	28.3	36.5	39.3	42.6	10.4	10.6	10.7	10.5	10.8	11
5	29	6.3	16.3	28.3	32.9	39.9	40	10.5	10.7	11.1	10.7	10.9	11.4
6	36	6.7	12.3	33.3	34	36.2	43.9	10.5	10.8	11.3	10.9	11.4	11.4
7	43	5.7	13.3	27	32.3	37.9	44.4	10.6	10.9	11.3	11	11.4	11.4
8	50	-	13.3	29.6	36.3	43.6	39.9	-	11.2	11.3	11.1	11.4	11.4
9	57	-	-	27	36.9	37.3	42.2	-	-	11.3	11.3	11.4	11.5
10	64	-	-	21.3	16.3	38.6	40.8	-	-	11.3	11.3	11.5	11.6

TABLE II: CPU and Memory Usage



(a) CPU Usage

(b) Memory Usage

Fig. 7: CPU and Memory Usage

VI. Conclusions and Future Work

This paper presents an SDN monitoring framework that uses OpenFlow protocol to estimate the data plane metrics reducing the load in CPU and switches. This work also proposes an orchestrator module that regulates the polling in switches based on the controller load and network size. Furthermore, this architecture uses profiling to separate the metrics to suit users requirements as well as the SM module facilitates the creation of different monitoring algorithms. The results of experiments using an open source OpenFlow controller and a network emulator to monitor the send data rate, loss rate and delay confirm the feasibility of the framework.

The future challenges comprise the improvement of the different monitoring algorithms including new metrics such as connection establishment time, jitter or packet duplication. Furthermore, the extension of monitoring tasks and metrics for other infrastructures based on SDN (Software Defined Wireless Mobile Networks, Software Defined Sensor Networks) can also be taken into account as well as the integration with other available monitoring tools. Similarly, the use of advanced analysis modules such as data mining, learning algorithms or traffic prediction can be taken into account to improve the management tasks developed in the control plane.

Acknowledgment

The research leading to these results has been partially funded by the European Union H2020 Program under the project SELFNET (671672). Ángel Leonardo Valdivieso Caraguay is supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program N° . 2543-2012.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, March 2008.

[2] A. L. Valdivieso Caraguay, L. I. Barona López, and L. J. García Villalba, "Evolution and Challenges of Software Defined Networking," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, pp. 1–7, November 2013.

[3] A. L. Valdivieso Caraguay, L. I. Barona López, A. Benito Peral, and L. J. García Villalba, "SDN: Evolution and Opportunities in the Development IoT Applications," *International Journal of Distributed Sensor Networks*, vol. 2014, May 2014.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)." RFC 1157 (Historic), May 1990.

[5] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)." RFC 6241 (Historic), June 2011.

[6] B. Claise, "RFC 3954 - Cisco Systems NetFlow Services Export Version 9." RFC 3954, October 2004.

[7] P. Phaal and M. Lavine, "Sflow version 5," July 2004.

[8] A. C. Myers, "JFlow: Practical Mostly-static Information Flow Control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, (New York, NY, USA), pp. 228–241, ACM, 1999.

[9] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks," May 2014.

[10] D. Raumer, L. Schwaighofer, and G. Carle, "MonSamp: A distributed SDN application for QoS monitoring," in *Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 961–968, IEEE, September 2014.

[11] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks," in *Network Operations and Management Symposium (NOMS)*, pp. 1–8, IEEE, May 2014.

[12] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Passive and Active Measurement*, pp. 201–210, Springer, January 2010.

[13] M. Shibuya, A. Tachibana, and T. Hasegawa, "Efficient Performance Diagnosis in OpenFlow Networks Based on Active Measurements," in *The Thirteenth International Conference on Networks ICN 2014*, pp. 268–273, February 2014.

[14] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.
[15] T. S. E. N. Zheng Cai, Alan L. Cox, "Maestro: A System for Scalable OpenFlow Control," tech. rep., December 2010.
[16] "Floodlight." http://www.projectfloodlight.org/.
[17] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, February 1993.
[18] M. Karl, J. Gruen, and T. Herfet, "Multimedia Optimized Routing in OpenFlow Networks," in *2013 19th IEEE International Conference on Networks (ICON)*, pp. 1–6, IEEE, December 2013.
[19] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for. Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, (New York, NY, USA), pp. 19:1–19:6, ACM, October 2010.
[20] "Highway." http://www2.tkn.tu-berlin.de/research/evalvid/qcif.html.
[21] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, (Berkeley, CA, USA), pp. 10–10, USENIX Association, April 2012.

Ángel Leonardo Valdivieso Caraguay was born in Loja, Ecuador, in 1985. He received a B.S. degree in Electronics and Telecommunications Engineering from the Escuela Politécnica Nacional, Quito, Ecuador in 2009 and a M.S. degree in Information Technology from the University of Applied Sciences Hochschule Mannheim, Germany in 2012. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His research interests include computer networks, software-defined networking and network function virtualization.

Jesús Antonio Puente Fernández was born in Madrid, Spain in 1988. Jesús received his Computer Science Engineering degree by Universidad Complutense de Madrid, Madrid, Spain in 2012. He received a M.S. degree in Computer Research in the Universidad Complutense of Madrid, Madrid, Spain in 2014. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His main research fields includes information networks and software defined networking.

Luis Javier García Villalba (SM'04) was born in Madrid, Spain, in 1969. He received the Telecommunication Engineering degree from the Universidad de Málaga, Spain, in 1993, the M.Sc. degree in computer networks in 1996, and the Ph.D. degree in computer science in 1999, the last two from the Universidad Politécnica de Madrid, Spain. He was a Visiting Scholar at the Research Group Computer Security and Industrial Cryptography (COSIC), Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium, in 2000, and a Visiting Scientist at the IBM Research Division (IBM Almaden Research Center, San Jose, CA) in 2001 and in 2002. He is currently an Associate Professor in the Department of Software Engineering and Artificial Intelligence at the Universidad Complutense de Madrid (UCM) and Head of the Complutense Research Group GASS (Group of Analysis, Security and Systems), which is located at the School of Computer Science at the UCM Campus. His professional experience includes research projects with Hitachi, IBM, Nokia, Safelayer Secure Communications, and VISA. His main research interests are in information security, computer networks, and software-defined networking. Dr. García Villalba is an Associate Editor in Computing for IEEE Latin America Transactions since 2004 and participates in the editorial board of several journals (IET Wireless Sensor Network, IET Networks, IJAHUC Journal, IJSIA Journal, IJFGCN Journal) and contributor of textbooks.

sensors

MDPI

Article

Monitoring and Discovery for Self-Organized Network Management in Virtualized and Software Defined Networks

Ángel Leonardo Valdivieso Caraguay and Luis Javier García Villalba *

Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), Faculty of Information Technology and Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases, 9, Ciudad Universitaria, 28040 Madrid, Spain; angevald@ucm.es

* Correspondence: javiergv@fdi.ucm.es; Tel. +34-91-394-7638

Academic Editor: Vittorio M. N. Passaro

Received: 27 December 2016; Accepted: 29 March 2017; Published: 31 March 2017

Abstract: This paper presents the Monitoring and Discovery Framework of the Self-Organized Network Management in Virtualized and Software Defined Networks SELFNET project. This design takes into account the scalability and flexibility requirements needed by 5G infrastructures. In this context, the present framework focuses on gathering and storing the information (low-level metrics) related to physical and virtual devices, cloud environments, flow metrics, SDN traffic and sensors. Similarly, it provides the monitoring data as a generic information source in order to allow the correlation and aggregation tasks. Our design enables the collection and storing of information provided by all the underlying SELFNET sublayers, including the dynamically onboarded and instantiated SDN/NFV Apps, also known as SELFNET sensors.

Keywords: 5G; Monitoring; NFV; SDN

1. Introduction

The management and customization of network services have been limited by the rigidity of traditional network architectures and increasing both capital and operational expenditures. Actually, the resolution of common traditional network problems, such as link failures, security attack, Quality of Service (QoS) or Quality of experience (QoE) degradation, bottlenecks, among others, requires the direct involvement of network operators. The manual re-configuration of the existing equipment or even the installation of new equipment (router, NATs, firewalls) compromises the normal operation of the network and causes the disruption of the Service Level Agreements (SLAs). Similarly, the creation of innovative value-added services is limited by the closed and proprietary hardware/software, and in some cases, all infrastructure may belong to the same provider.

Those limitations make traditional network architectures unfeasible to meet the requirements of today's users, enterprises and carriers. The solution proposed to solve these challenges is following the advances reached by computing, where developers can create their own applications using a high level programming language. The programs can be executed with different equipment thanks to the abstractions of resources provided by the Operating Systems. In this context, Software Defined Networking (SDN) and Network Function Virtualization (NFV) appears as a promising strategy to reach those objectives. SDN proposes the decoupling of data and control planes in network devices enabling their independent development and a centralized view of the network. NFV promotes the migration from typical network equipment (DPI, firewall, load balancers) to a software packages or network functions (NF) that can be instantiated in a virtualized infrastructure. Both architectures

are complementary and potentially could be integrated to provide an open network environment for developers.

Furthermore, the exponential growth of mobile devices and content together with the advent of cloud services bring additional challenges to operators and service providers. A radical decrease of integrated network management operations without negatively affecting the QoS/QoE and security is required. Similarly, a new model that integrates the access and management of mobile resources is promoted. The future 5G architecture is expected to expose not only typical mobile broadband but also a heterogeneous, simplified and unified control [1]. The network management expenditures can be reduced through automation of operations. In this context, a scalable management framework that includes data mining, pattern recognition, learning algorithms to reduce operation expenditures is challenging.

The present paper comprises the following contributions. Firstly, the Self-Organized Network Management in Virtualized and Software Defined Networks Project (SELFNET) [2] is described. SELFNET uses SDN/NFV principles to provide smart autonomic management of network functions in order to resolve network problems or improve the QoS/QoE. SELFNET integrates the self-management paradigm with the use of data mining, learning algorithms, pattern recognition to identify the network behaviour and including 5G mobile architectures. The SELFNET architecture is composed of well defined layers: Infrastructure, Virtualized Network, SON Control, SON Autonomic and Access Layer.

Secondly, the present paper proposes the SELFNET Monitoring and Discovery Framework, as part of the Monitor and Analyzer sublayer of the SON Autonomic Layer. The SELFNET Monitoring and Discovery Framework lay the SDN/NFV foundations to provide a contextaware model based on a customizable set of low-level metrics. In contrast with traditional network monitoring schemas, the present framework integrates different sources within a single customizable system. In this way, it is able to organize the collected information according to different criteria such as virtual instances by tenant, metrics by virtual instances, physical devices by network location, physical metrics, flow statistics by device and metrics by sensor. Similarly, the proposed framework will facilitate not only the querying process of gathered information but also the management of large amounts of data from heterogeneous data sources. In this way, the data of each of these layers can be correlated to provide enhanced information of the network status such as the relation between virtual instances and their respective physical device, the physical and virtual instances where a sensor is running, information related to LTE devices, edges and locations. Through it, the administrator is able to establish policies based on high level paradigms or Health of Network (HoN) metrics. This resulted in a significant reduction in the costs and improvements to data quality and timeliness.

The rest of the paper is outlined as follows: Section 2 describes SDN, NFV and the related works focused on management and monitoring. Then, Section 3 explains the SELFNET Framework. Section 4 defines the Monitoring and Discovery Framework. Section 5 describes the sensor workflows (onboarding, instantiation and monitoring) following by each component of this framework. Section 6 describes the implementation of the framework. Finally, Section 6 presents the conclusions and future work.

2. Software Defined Networking and Network Function Virtualization

2.1. Software Defined Networking

Software Defined Networking (SDN) was born due to the lack of flexibility and programmability issues of the traditional deployments. The updating/change in network behaviour requires the manual reconfiguration (one by one) of the equipment. The inclusion of new network services is delayed due the large standardization process to introduce a new technology or standard [3]. SDN [4] proposes the separation between data and control plane in network devices enabling their independent development. It also proposes the logical centralized control of the network devices by means of an element known as controller. This element has a whole view of the network status and can manage all

physical devices. The controller abstracts the network resources and provides an API that can be used by the network administrator to develop the network services.

SDN is promoted by the Open Networking Foundation (ONF) [5], which is a non-profit organization supported by the network operators and service providers. The SDN architecture defines three layers: Data layer, Control layer and Application layer in order to expose the network resources to external applications, as shown in Figure 1.

- Data layer: this layer exposes the resources of hardware devices (switches or routers) towards the control layer. The communication between data and control layer is done by the southbound interface.
- Control layer: the controller manages the network devices based on its policies or in high level applications. The API between control and application layer is the northbound interface.
- Application layer: this layer allows the development of high level applications in order to customize easily the network behaviour.

The main southbound interface is the OpenFlow protocol [6]. In contrast to well-known protocols based on traditional network architectures (NetFlow, sFlow, SNMP), OpenFlow is the first standard interface for the communication between data and control planes in SDN architecture. In this context, an OpenFlow switch takes into account the common set of functions of traditional switches in order to forward the information based on the controller instructions. The control plane (controller) sends OpenFlow messages to modify the rules in the switch tables and control the handling of the incoming packets.

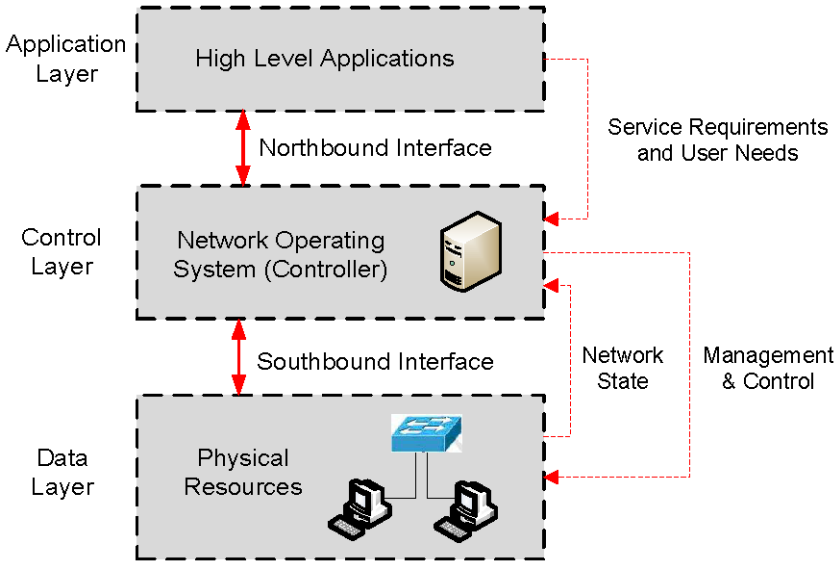


Figure 1. SDN Architecture.

2.2. Network Function Virtualization

The term "Virtualization" is not a new concept. The virtualization refers to the abstraction of the logical resources from the physical resources, creating multiple logical instances over the same physical infrastructure [7]. The virtualization has reached different technical domains: virtualization of operating systems, hardware platforms, storage capacities and networks. In the networking field, the virtualization enables the sharing of multiple virtual networks (VNs) over a physical network simultaneously. However, the virtualization principles have also been extended to other functionalities of networks.

Nowadays, the telecom providers invest a lot of money updating or installing new network functions (NFs) or appliances. Often, the appliances (e.g., firewall, DPI) are deployed in proprietary

hardware or private software, and consequently, cannot be reused or modified by other service providers. Moreover, the rigidity and complexity of network deployments reduce the customization capabilities of the services. In this context, the Network Function Virtualization (NFV) [8,9] proposes the transferring of the Network Functions NF (routing, firewall, deep packet inspection DPI, gateway) towards virtual software-based applications, which are executed in IT platforms (servers, switches and storage). This new vision of IT services provides a major flexibility/scalability and facilitates the development of new Virtual Network Functions (VNFs), while decreasing costs. Figure 2 describes the differences between NFV and traditional architectures.

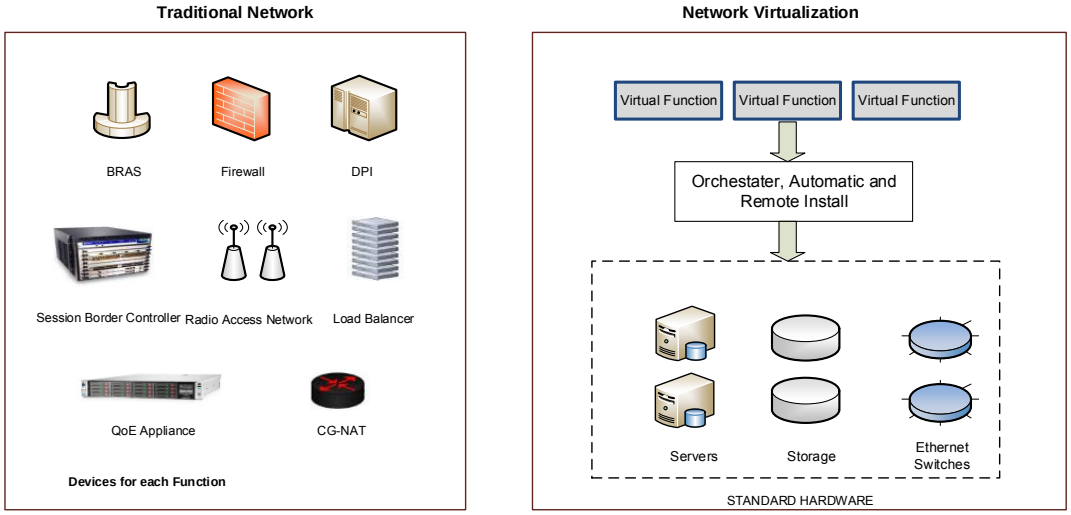


Figure 2. Traditional Network Functions vs. Virtual Network Functions.

The European Telecommunications Standards Institute (ETSI) has released the first NFV specification [10]. The NFV architecture is composed of three main modules: NFV Infrastructure (NFVI), NFV and NFV Management and Orchestration (NFV M&O), as is shown in Figure 3.

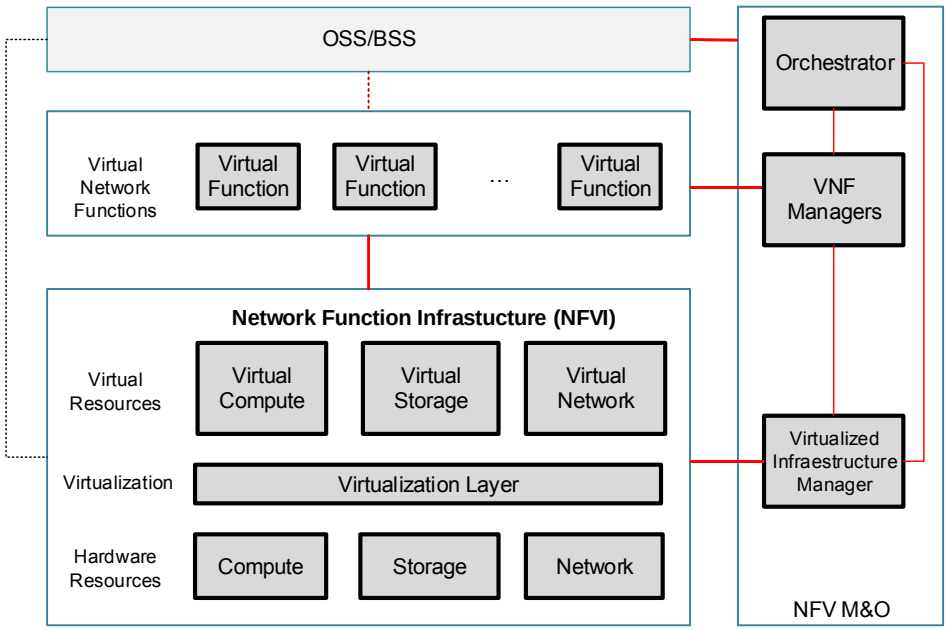


Figure 3. NFV architecture.

The NFV components are described below:

- NFVI: It represents the hardware and software resources of the system where the NFV concept are applied. Computing, storage and networking resources are included. The virtualization layer abstracts the different resources and enables the isolation and independence of virtual compute, virtual storage and virtual network for different tenants.
- VNF: It represents a virtual NF instance that runs over the NFVI. A virtual NF could be deployed in a single virtual machine (VM) as well as over multiple VMs (different components in each virtual machine).
- NFV M&O: The main function is the orchestration and management of the VNF and NFVI ensuring the optimal and effective operation of the VNFs in the available infrastructure. The principal NFV M&O components are: the orchestrator, the VNF manager and the Virtualized Infrastructure Manager (VIM). The VIM uses a resource inventory to control the resources availability, which guarantees the provisioning of services.

2.3. Related Works

The SDN and NFV paradigms have been applied in several research areas. However, the present work focuses on two main research topics: network management and monitoring. In the first area, the use of SDN/NFV in the development of novel management architectures over virtualized environments has gained the interest of several consortiums formed by operators, academia and service providers. In Table 1, relevant SDN/NFV-based management projects are described. These projects involve different application scenarios. However, the provisioning of autonomic management operations capable to provide a scalable, extensible and smart network management is at the beginning stage of research.

Table 1. SDN/NFV based management ongoing projects.

Project	Domain	Description	Application Scenario
CROWD [11]	• SDN • SON	The project aims to bringing density proportional capacity in heterogeneous wireless access networks. Similarly, it focuses on guaranteeing mobile user's QoE, optimising MAC mechanisms and proportional energy consumption. In this way, it enhances the traffic management in dense wireless networks.	• Traffic Management
5G-NORMA [12]	• SDN • NFV	The project focuses on providing adaptability of a resource in an efficient way. The framework handles fluctuations in traffic demand resulting from heterogeneous and dynamically changing service portfolio. The novel network functions offer resource-efficient support of varying scenarios and help to increase energy-efficiency.	• Multi-service scenario • Multi-tenancy scenario
MCN [13]	• SDN	The project focuses on the enhancement of traffic processing by means of the separation between radio hardware and packet forwarding hardware.	• SDN environment
UNIFY [14]	• SDN • NFV	The project aims to develop an automated, dynamic service creation platform through the creation of a service model and service reaction language. It enables the dynamic and automatic placement of networking, computing and storage components across the infrastructure. Similarly, the orchestrator includes optimization algorithms to ensure optimal placement of elementary service components across the infrastructure.	• Infrastructure Virtualization • Flexible Service Chaining • Network Service Chain Invocation for Providers
T-NOVA [15]	• SDN • NFV	The project focuses on the deployment on Network Functions-as-a-Service (NFaaS) over virtualised Network/IT infrastructures. For this purpose, it designs and implements a management/orchestration platform for the automated provision, configuration, monitoring and optimization of virtualized resources. Moreover, SDN is also used for efficient management of the network infrastructure.	• High-Level Scenario • VNF Channing Scenario

Regarding the second research area, the monitoring on SDN and NFV plays a major role to the good operation of the network. The decisions taken by the different services depends on the high-speed and the accuracy of the information provided by monitoring services. In this context, the work proposed in [16] uses SDN to virtualize a switch to monitor traffic without the need of port mirroring. However, the system does not collect additional metrics at different levels (e.g., virtual, sensors). Similarly, NetAnlytics [17] can efficiently monitor data plane packet flows and uses NFV to instantiate the network functions. Nevertheless, the collected data is focused on flow metrics and does not provide customization of heterogeneous sources.

The monitoring framework presented on [18] enables the Virtual Infrastructure Manager monitoring. It includes different agents able to collect data from VNFs. However, the monitoring framework is considered to be an internal functional block of the VIM and does not include additional sources (e.g., physical metrics) and customization of tasks (e.g., aggregation). In [19], the authors propose a reference framework for traffic engineering for SDN networks. The framework is composed of traffic measurement (network level) and traffic management (QoS, load balancing). However, the proposal does not include the customization capabilities provided by NFV engine.

3. Self-Organized Network Management for SDN/NFV (SELFNET)

The SELFNET H2020 project focuses on design and implement an autonomic network management framework to provide Self-Organizing Network (SON) capabilities in new 5G mobile network infrastructures. By automatically detecting and mitigating a range of common network problems, currently manually addressed by network administrators, SELFNET aims to provide a framework that can significantly reduce operational costs and consequently improve the user experience [2,20]. By exploring the integration of novel technologies such as SDN, NFV, SON, Cloud Computing, Artificial Intelligence, QoS/QoE and next generation of networking, SELFNET will provide a scalable, extensible and smart network management system. The framework will assist network operators to perform key management tasks such as automatic deployment of SDN/NFV applications that provide automated network monitoring and autonomic network maintenance delivered by defining high-level tactical measures and enabling autonomic corrective and preventive actions to mitigate existing or potential network problems.

SELFNET addresses three major network management concerns by providing self-protection capabilities against distributed network attacks, self-healing capabilities against network failures, and self-optimization features to improve dynamically the performance of the network and the QoE of the users. The facilities provided by SELFNET will provide the foundations for delivering some of the 5G requirements defined by 5G-PPP consortium. In this context, the Figure 4 illustrates the architecture of the SELFNET framework. The architecture is based on five differentiated layers with the following logical scopes: Infrastructure Layer, Virtualized Network Layer, SON Control Layer, SON Autonomic Layer, NFV Orchestration & Management Layer, SON Access Layer.

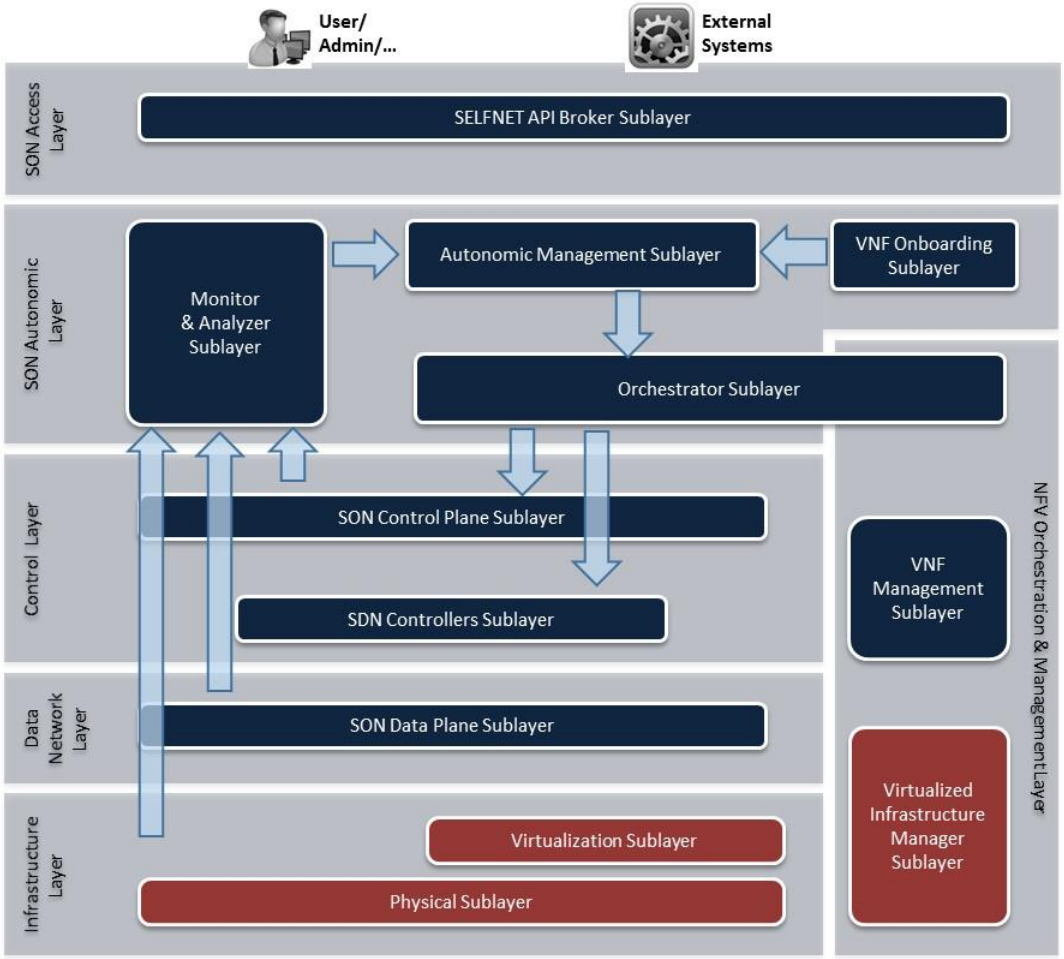


Figure 4. SELFNET Architecture Overview.

3.1. Infrastructure Layer

This layer provides the resources required for the instantiation of virtual functions (Compute, Network and Storage) and supports the mechanisms for that instantiation. It represents the NFVI as defined by the ETSI NFV terminology [10]. In order to achieve its functionality, two sublayers will be defined: Physical Sublayer and Virtualization Sublayer.

- ➢ Physical Sublayer: It includes the physical resources required to provide computation, networking and storage capabilities over bare metal. Since SELFNET is also designed to include 5G networks, the physical elements follows a mobile edge architecture in which operators can deploy the operational and management services. The framework follows the Mobile Edge Computing (MEC) architecture proposed by ETSI [21] where the edge nodes are geographically separated from the data centre.
- ➢ Virtualization Sublayer: It enables the sharing of the available resources between different users or services. It offers several advantages, such as the isolation, reliability, adaptability and control of resources. However, the main disadvantage includes the poor performance in the virtualization tasks. Regarding this research topic, recent advances on virtualization technologies have reached the expectations on 5G infrastructures on the performance of virtualized workload with intensive Input/Output (I/O) [22]. In other words, the performance penalty of using virtualization can be considered as negligible for modern devices. In SELFNET, the virtualization sublayers includes the use of virtual switches used to connect Virtual Machines allocated on the physical resources.

3.2. Data Network Layer

In this layer, the different functionalities of the networks functions are located and interconnected in a designed topology. The NFs include the instances required for the normal operation of the virtual infrastructure and those created by SELFNET as part of the SON functionalities. The Data Network Layer also provides multi-tenancy support [23]. Multitenancy enables the sharing of resources among different tenants, each with their own administrative domain and business requirements. In this context, the term Software Defined Networking plays an important role. In traditional architectures, the network elements is composed by a specialized hardware in the packet processing (data plane), and over the hardware works an operating system (e.g., Linux) that receives information from the hardware and executes a software application (control plane). Instead, SELFNET follows the SDN principles and proposes the separation and decoupling of Data Network Layer and SON Control Layer. Additionally, open interfaces between them are established. Therefore, OpenFlow [6] is the most widely used protocol for exchanging information between data and control planes. The advantage of OpenFlow is the use of available elements and features of traditional network hardware. OpenFlow opens up these elements with a firmware update avoiding a complete change of hardware. In this way, OpenFlow acts as a standard way of conveying flow-table information to the network devices, so these can be controlled externally.

3.3. SON Control Layer

This layer includes the elements responsible of collecting data from different virtualized sources (SON Sensors) and the functions that execute actions into the network (SON Actuators). The SON Sensors and SON Actuators are controlled by the SON Autonomic Layer, which provides network intelligence. Similarly, the SON Control Layer deals with the control plane in SDN architectures. In other words, it translates autonomic network-wide policies into specific network element configurations. This layer is composed of two sublayers: SDN Controller Sublayer and SON Control Plane Sublayer.

- SDN Controller Sublayer: It implements a logically centralized controller (e.g., SDN control plane) and provides the governance of the network elements and the control of the network functions. The SDN controller uses the information of the network behaviour and enforces the execution of the rules in the network elements. In this way, the traffic passing through such network elements can be dynamically modified. For this purpose, the SDN controller uses a well-defined API and standardized forwarding instruction set (e.g., OpenFlow).
- SON Control Plane Sublayer: It instantiates the different network functions (NF) in the virtualized infrastructure. In SELFNET architecture, which aims to provide self-organized capabilities, there are two types of NFs: SON Sensors and SON Actuators. The SON Sensors collect data related to network activities. The collected information includes metrics related to global traffic (e.g., link status, bandwidth) or specific metrics (e.g., DPI, QoS on a video streaming of a specific data flow). The SON Actuators execute a specific set of actions on the traffic circulating in the network. The actions depend on the application developed by the service providers. For instance, if the system detects a DDoS attack, a SON actuator can automatically block the specific attack source. If the system detects a QoS degradation, another SON Actuator can optimize the network flow increasing the priority or bandwidth.

3.4. SON Autonomic Layer

This layer is responsible of providing the network intelligence. The information collected from sensors is used to diagnose the network situation. Then, the actions to accomplish the systems goals are determined and executed. The main components of SON Autonomic Layer are described as follows.

3.4.1. Monitor and Analyzer

This sublayer collects the information provided by sensors. Then, this information is aggregated and correlated in order to extract the relevant information. The analyser uses the relevant information to detect network situations (botnet detected, QoS/QoE degradation, DDoS attack, link failure). The whole process is organized in three steps: Monitoring and Discovery, Aggregation and Correlation and Analyzer.

- Monitoring and Discovery. It collects the data sent by the SON Sensors. For this purpose, when a new Sensor is instantiated, it receives the notification and instantiation details and establishes a connection in order to receive the corresponding metrics. Moreover, it also receives the information provided by the physical and virtual sublayers. Then, the information is stored in a database in order to be processed by upper layers.
- Aggregation and Correlation. It performs the correlation and aggregation of the information stored in the monitoring and discovery database. This process involves additional actions, such as the data normalization, verification and removal of redundant information. At the end of this stage, only relevant information will be processed by the Analyzer module.
- Analyzer. Its main purpose is the comprehensive analysis of the relevant information provided by Aggregation and Correlation. The analysis also includes the prediction of future network problems. The network problems are known as Health of Network (HoN) due the global vision or system-level scope of the analysis. For this purpose, it takes advantage of several prediction, pattern recognition algorithms and big data techniques. The trending and predicted values for the metrics enable the application of proactive and reactive actions in the system. At the end of this stage, the network events are sent to the autonomic manager in order to establish the corresponding actions in the network.

3.4.2. VNF Onboarding

It acts as a repository of the different network functions NFs. In this sublayer, the available network functions are stored and their capabilities are disseminated to the other sublayers. Similarly, the service providers can design, create and update their own applications. In this context, the encapsulation of NFs follows the recommendations of the ETSI MANO framework for the NFV [24]. Consequently, the VNF Manager (VNFM) is the key component for the lifecycle of SON sensors and actuators. The VNFM lifecycle exposes a common set of primitives for the automated instantiation, configuration, re-configuration and termination of the different VNFs. A common API enables service providers the easy design and development of their solutions. Once a solution (NFs) is onboarded, the autonomic manager can use their capabilities to provide the new service (sensor/actuator).

3.4.3. Autonomic Manager

It uses different algorithms to diagnose the root cause of a network problem in terms of the HoN metrics provided by the Analyzer. Once the cause is detected, the autonomic manager uses the available NFs provided by the VNF onboarding to decide the best reaction strategy or a countermeasure (e.g., deploy a new balancer, firewall or DPI). Then, the taken actions are sent to the NFV Orchestration and Management Layer. The related tasks are organized in three well defined modules.

- Diagnoser. It diagnoses the root cause of the network situations notified by the analyzer. For this purpose, it uses the information available on Monitor & Analyzer sublayer (topology, sensor data, HoN metrics) and takes advantage of stochastic algorithms [25,26], artificial intelligence [27,28], data mining [29] to estimate the location of the source of the problem. Then, the root cause is notified to the Decision Maker. Because Diagnoser follows the extensibility and customization capabilities defined in 5G networks [1], the architecture enables the existence of a dataset of several algorithms. Depending of the complexity of the situation, SELFNET defines three different

diagnosis processing chains: TAL-based diagnosis, machine learning diagnosis and offline diagnosis. More information about diagnose component can be found in [30,31].

- ➢ Decision Maker. It takes the incoming diagnosis information and decides a set of reactive and proactive actions to be taken into the network in order to avoid the detected and emerging network problems, respectively. Similarly, it also takes advantage of the integration of artificial intelligence algorithms to determine the responses or tactics to be taken. The taken decisions are notified to the action enforcer.
- ➢ Action Enforcer. It provides a consistent and coherent scheduled set of actions to be taken in the infrastructure. In other words, it validates, organizes and refines the tactics to avoid conflicts, duplications and nonsense order of actions. At the end of this stage, a high level description of the location, type of SELFNET SON Actuators, related configuration parameters are transferred to the orchestrator.

3.5. NFV Orchestration and Management Layer

This layer is responsible for the control and chaining of the different NFs in the virtualized infrastructure. The architecture follows the ETSI MANO [24] recommendations and, consequently, it is composed of: Orchestration, VNF Management and Virtualized Infrastructure Management (VIM). As described in Section 3.4.2, the VNF Management operations are partially developed in the VNF Onboarding. The other operations are described as follows:

- ➢ NFV Management and Orchestration. It is responsible for receiving the set of actions of the Autonomic Manager and orchestrate the network functions in the available virtual resources. The coordination and schedule of the enforcement of different actions is executed by the interaction with the virtual infrastructure manager.
- ➢ Virtual Infrastructure Manager VIM. It is responsible for organizing and providing the virtual resources for the instantiation of the different network functions. The VIM interacts with the physical and virtual infrastructure to ensure the availability of resources and perform the automatic deployment of services.

3.6. SON Access Layer

This layer provides an appealing and intuitive interface that enables different monitor and operation capabilities depending on the authorized users. In this way, the users can check the current health status of the SELFNET operations. Similarly, the Access API lists the SON Sensor and Actuator currently deployed in SELFNET as well as the loggings and messages in order to enable a wider view of the SELFNET status. This interface is used by external actors such as Business Support Systems (BSS) or Operational Support Systems (OSS). As described in the previous sections, SELFNET aims to be an independent and autonomous solution that acts mitigating or solving network problems without any actions from real users. In this way, the SON Access Layer also provides users the validation of the actions taken by the SELFNET autonomous system.

4. SELFNET Monitoring and Discovery

One of the main challenges of the SELFNET infrastructure is the monitoring and discovery of the different metrics generated by the underlying virtualized infrastructure. The metrics collected can include low-level metrics, Key Performance Indicators (KPI) and Health of Network Metrics (HoN) in order to have a complete knowledge about the network status. However, unlike the traditional monitoring solutions, where the monitoring nodes and information provided are static, the SELFNET sensors are virtualized network functions that can be dynamically allocated in different sections of the network. Moreover, the SELFNET Monitoring and Discovery Framework should be able to monitor a large amount of low-level metrics from several data sources. In this context, Figure 5 shows the eight interfaces used for either collecting metrics or sending the results of analysis tasks.

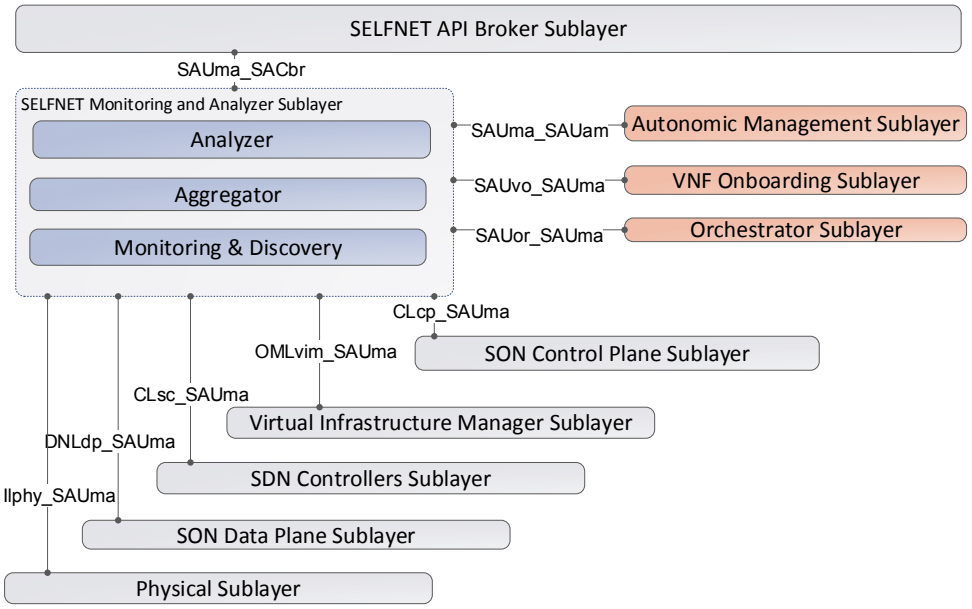


Figure 5. Monitor and Analyzer Sublayer Interfaces.

The SELFNET Monitor and Analyzer interacts with other sublayers through different APIs. The Table 2 describes the interface name, source, destination and the provided information. On one hand, several sources send their corresponding information to be processed and analysed. On the other hand, the output of the analysis is sent to the autonomic sublayer and the status of the sublayer is delivered to the broker sublayer. In order to achieve the functionality established by the SELFNET architecture, the proposed framework takes into account different design principles and methodologies, as summarized in Table 3. A detailed description of the SELFNET System Requirements can be found in [30].

Table 2. Monitor and Analyzer Interfaces.

Interface Name	Source	Destination	Information
ILphy_SAUma	Physical	Monitor and Analyzer	Physical metrics
OMLvim_SAUma	Virtualized Infrastructure Manager (VIM)	Monitor and Analyzer	Virtual resources and metrics
DNLdp_SAUma	SON Data Plane	Monitor and Analyzer	Data Plane metrics
CLsc_SAUma	SDN Controllers	Monitor and Analyzer	SDN Controller metrics
SAUvo_SAUma	VNF Onboarding	Monitor and Analyzer	NFV Sensors description
SAUor_SAUma	Orchestrator	Monitor and Analyzer	Instantiation of NFV Sensors
SAUma_SAUam	Monitor and Analyzer	Autonomic Management	High level summary of actual and predictive network issues
SAUma_SACbr	Monitor and Analyzer	API Broker	Status of Monitoring and Analyzer

Table 3. Framework architectural requirements.

Requirement	Description
Layered architecture	The framework follows a layered architecture, including a number of ordered and logically separated layers. In this way, the complexity on development process (design, implementation, evaluation) is reduced. Similarly, the interoperability between different vendors and technologies is supported.
Extensibility/Flexibility	The different layers and sublayers are composed using a modular design. The framework modularity offers advantages in terms of the usefulness in extensibility/augmentation. The open interfaces and APIs can be optimized by third parties to adapt their own solutions. Similarly, additional functionalities can be permanently or temporally integrated.
Multi-level scalability	The framework addresses network management concerns in large-scale networks. The monitoring sources are spread over virtualized network elements located strategically at regional or global levels. The framework enables elastic scalability employing the cloud computing engine.
Standards compliance	The design of the framework adheres to the standards that are considered relevant to the domain. It includes ETSI standards for NFV [24] and Open Networking Foundation (ONF) standards for SDN [5].

4.1. High Level Architecture

The main goal of Monitoring and Analyzer sublayer is to provide a consistent set of components to analyze network status by the proper use of the monitoring information gathered from the running network infrastructure. With this objective in mind, the Monitoring and Analyzer Sublayer was divided in three sections: Monitoring and Discovery, Aggregation and Analyzer. The Analyzer Framework relies on Health of Network (HoN) metrics to determine network status. The outcomes of this framework are sent to the Autonomic Management Sublayer. HoN metrics, in turn, are derived from the aggregated or correlated low-level metrics and events provided by the Aggregation Framework. Low-level metrics are collected in the Monitoring and Discovery Framework.

The whole architecture is divided in functional blocks, as shown in the Figure 6. In the bottom part, the underlying data sources are located. The Monitoring and Discovery framework introduces the concept of Data Source as a functional component that allows data to be transferred from the corresponding monitored element to the framework. A Data Source is capable to choose the communication method, either polling or pushing, to gather or receive data from the monitored elements.

Polling and Pushing are main approaches used for distributed information dissemination [32,33]. The pull model is based on the request and response paradigm. In other words, the user initiates the data transfer and requests to the server for a specific piece of information, either synchronously or asynchronously. The server receives the request and responses back to the initiator with the requested information. Examples of the pushing approach include SNMP [34] or HTTP [35]. Meanwhile, the pushing method is based on the publish/subscribe/distribute paradigm. In this case, each agent (publisher) individually takes the initiative and sends the information to the subscribers through a distributed engine. The subscribers listen the set of channels based on their individual interest. Examples of pushing approaches include GCM [36], XMPP [37].

A Data Source implements the required interface to communicate with the monitored element. For example, in order to receive data from a particular sensor, the Data Source may implement a server-side REST interface with a particular subset of methods exposed to the sensors, letting them send data to the monitoring framework. The Discovery function is related with the framework capabilities to detect the instance of new data sources and communicate them in order to collect metrics.

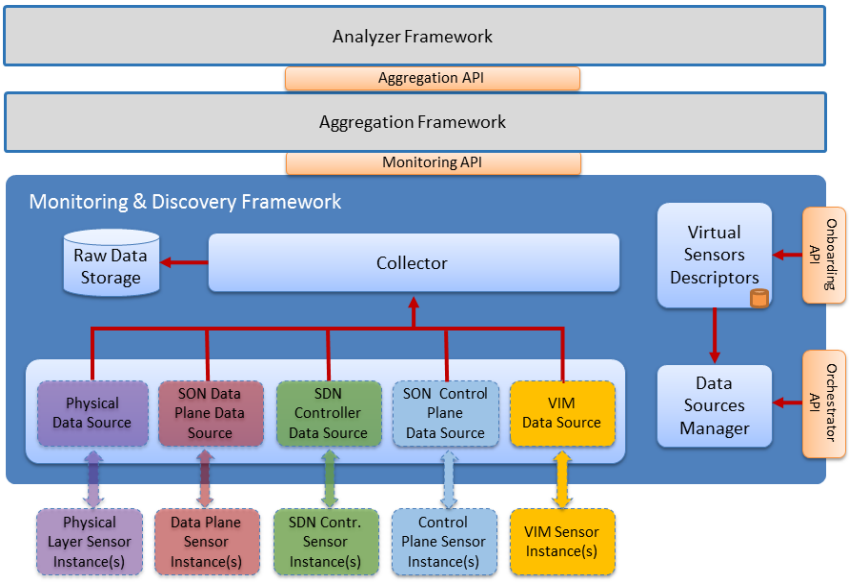


Figure 6. Monitoring and Discovery Framework Architecture.

4.2. Virtual Sensors Descriptor

The Virtual Sensors Descriptor component is responsible to receive, parse and store the information regarding onboarded sensor types available on SELFNET catalogue. This element interacts with two other SELFNET components: VNF Onboarding Sublayer and Data Sources Manager.

The Virtual Sensors Descriptor uses the SAUvo_SAUma interface to allow the communication with the VNF Onboarding Sublayer. This interface uses a communication channel allowing VNF Onboarding Sublayer to report to Virtual Sensors Descriptors component whenever a sensor is onboarded, updated or removed in SELFNET catalogue. Every time an operation is performed in the SELFNET catalogue, the Virtual Sensor Descriptors need to be notified in order to receive this information, parse it and update its internal sensor type database.

The VNF Onboarding Sublayer provides a message queue to broadcast any action performed on the catalogue, the Virtual Sensors Descriptors have a client that subscribes this channel and updates the local sensor catalogue. The local sensor catalogue is available to the Monitor and Discovery framework via Virtual Sensor Descriptors—Data Sources Manager interface. This catalogue mirrors, from the VNF Onboarding sublayer main catalogue, the essential sensor information needed to connect to a sensor and retrieve the sensed data. The catalogue does not have information regarding running sensor instances, but provides data related to the sensor metrics and communication.

4.3. Data Sources Manager

In the SELFNET monitoring framework, the Data Source is defined as a functional component in the monitor side, which serves as an interface between the monitor and the monitored element and is in charge of collecting data from the corresponding monitored device. For the instantiation of the data sources, the Data Sources Manager requires specific information (e.g., communication protocol, poll interval) about the sensor, which is obtained during the sensor onboarding phase. As soon as the sensor is instantiated, the Data Sources Manager is notified by the Orchestrator with the relevant instantiation information (e.g., IP address) from the sensor in order to correctly configure the data source. At this stage, the data source is ready to start collecting data.

4.4. Data Sources Instances

This section describes the different data sources and the way that the metrics are collected. As depicted in the Figure 7, the data collection shares a common structure. The source instances send

the metrics to the corresponding data source. Then, the information is stored in the raw data through the collector.

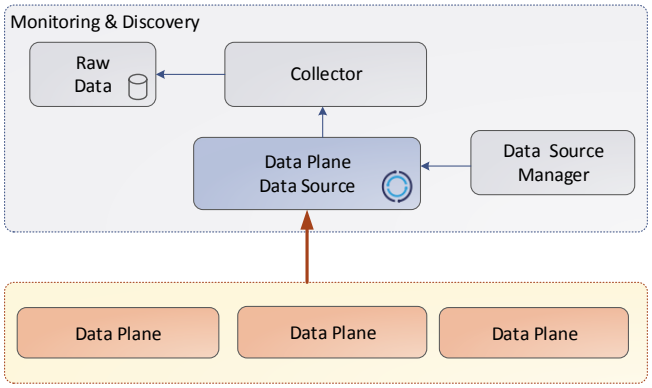


Figure 7. Data Source.

Table 4 summarizes the particular types of metrics depending on the source instances.

Table 4. Data Source Instances.

Data Source Instance	Description
Physical Infrastructure	Metrics related with the physical equipment, which holds the virtual resources. It also includes the network elements that enable connectivity between the different components.
Virtual Infrastructure Manager	Metrics about the virtualized resources from the cloud environment that are running on top of the physical sublayer. It includes the tenants, virtual switching and routing management, and the location of compute resources attached to switch ports.
SDN Controller	It provides the information related with the network behaviour based on the view of the SDN Controller. In other words, the controller can infer the network statistics based on the information sent by the control plane—data plane interface (e.g., OpenFlow).
SON Data Plane	It provides low level network traffic characteristics. In this context, the flow level monitoring has become an appropriate solution. Flow level measurements can provide useful traffic statistics with small amounts of measured data.
SON Control Plane	It provides the collection of metrics from sensors that will be deployed through the entire virtual infrastructure. These sensors measure and collect specific physical and/or virtual metrics depending on the purpose of the sensor.

4.5. Collector

In the SELFNET monitoring framework, the Collector is defined as a functional component in the monitor side, which listens, on a message bus, for Data Sources metrics and events, transform and publish data into a Database and External Consumer Systems. The Collector uses a pluggable storage system, meaning that it can be changed by other databases if needed. This function occurs in multiple pipelines associated with Data Sources in order to provide configurable data flows for transformation and publication of data.

4.6. Raw Data Storage

Storing Metrics and Events offers different challenges. Metrics have a regular behaviour and events have an unpredictable nature. SELFNET will use more than one kind of database to store events

and metrics. Metrics will be stored in a TSDB (Time Series database). These types of databases are optimized for handling time series data, arrays of numbers indexed by time (a datetime or a datetime range). Events will be stored in a NoSQL database whose main purpose is to store unstructured data for a short period of time. Raw data storage has the main goal to provide a stage where data can be queried for a short period of time. In addition, this storage can be used for more detailed analysis in order to take better decisions. It supports two kinds of policies for metrics that need to be described on sensor descriptors. As an example, policies should look like: 1 day timespan for 1 min granularity and 7 days timespan for 2 min granularity. This time aggregation is not intended to overlap features of the Aggregation Layer, it is only a way to support raw data for longer periods.

4.7. Northbound API

The Monitoring and Discovery Framework exposes several APIs for other SELFNET components, but mainly for the Aggregation Framework to retrieve the available data. All stored data is published to a message bus, included in the Monitoring API, and can be reached by other SELFNET consumers. This should be the preferred method for forwarding data to the Aggregation layer.

The information stored in the Raw Database can be accessed by a REST API. This API supports authentication and enable multiple operations: listing all metrics created, retrieving measures and filtering them over a time range, among others. It is also possible to query data for a specific granularity and for all available granularities. Optionally, data related to Events is stored in a Raw Database, but the preferred method to consume it is from message bus without any treatment or association logic. The database also exposes a REST API in order to query events that occurred in a near past.

5. Monitoring and Discovery Workflows

This section highlights the workflows across different components of Monitoring Framework and their relation with external components like the Onboarding or Orchestrator Sublayer.

5.1. Sensors Onboarding

Whenever a new sensor is onboarded to the architecture, several steps are carried out in order to inform other SELFNET components about the sensor functionalities and descriptors. This process is known as sensor onboarding and is depicted in Figure 8.

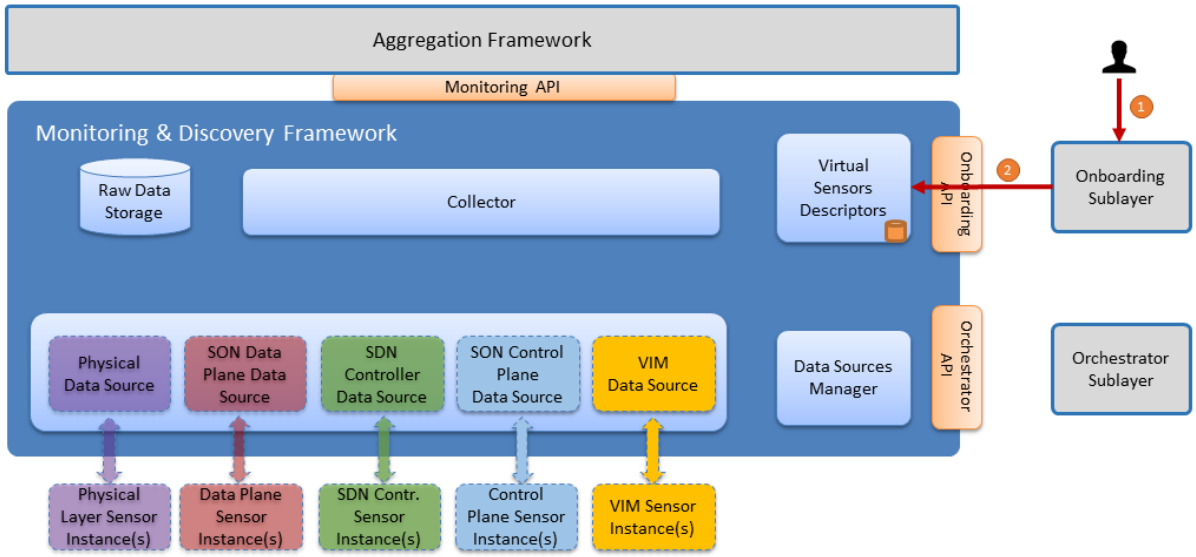


Figure 8. Sensor Onboarding Workflow.

Two main components participate in this workflow, the Onboarding Sublayer and Virtual Sensors Descriptors, as is described in Table 5.

Table 5. Sensor Onboarding Workflow Steps.

Step	Source Component	Destination Component	Description
1	API Broker Sublayer	Onboarding Sublayer	A new virtual sensor is onboarded to the system either by using the SELFNET Graphical User Interface (GUI) or by an external system. Further details about the virtual sensor onboarding process can be found in Deliverable 3.1 [38].
2	Onboarding Sublayer	Virtual Sensors Descriptors	The onboarding of a new sensor is published by the Onboarding Sublayer to a message bus to which the framework is subscribed by means of the Onboarding API. The Virtual Sensors Descriptors takes care of collecting all the information needed to build its own representation of the sensor.

5.2. Sensor Instantiation

Sensor instantiation has a direct impact in Monitoring and Discovery Framework. Once instantiated, a sensor will start to produce data that will subsequently be available to the Monitoring and Discovery framework. For an effective communication between the sensor and the framework, a valid Data Source is required. Sensor instantiation workflow is depicted in Figure 9.

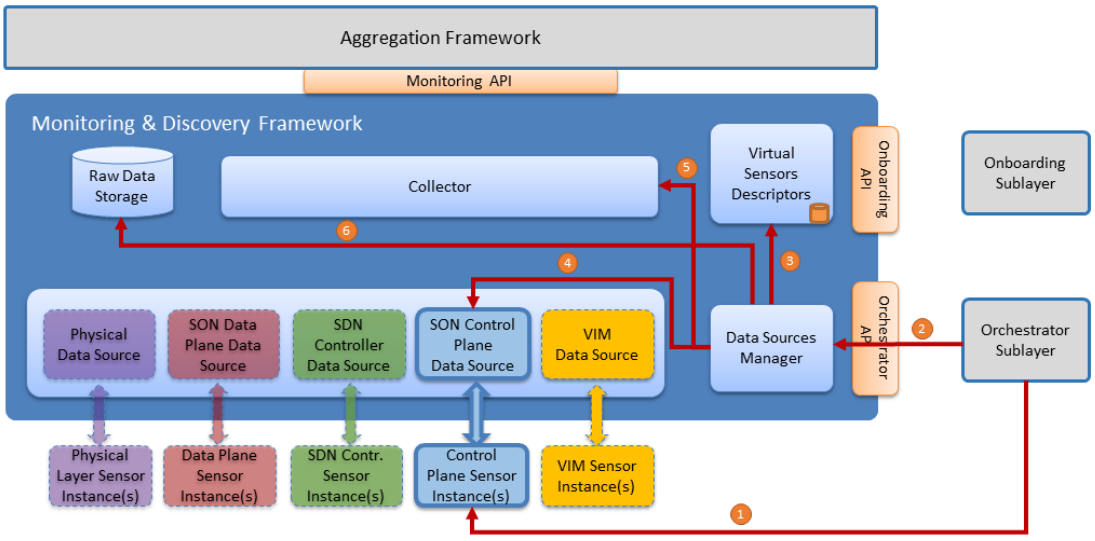


Figure 9. Sensor Instantiation Workflow.

This workflow is described in Table 6 and it takes into account a Control Plane Sensor Instance. It is important to note that all sensors from the five sources follow the same process.

Table 6. Sensor Instantiation Workflow Steps.

Step	Source Component	Destination Component	Description
1	Orchestrator Sublayer	Sensor Instance	A new virtual sensor (e.g., Control Plane) is instantiated by the Orchestrator Sublayer (using the VIM).
2	Orchestrator Sublayer	Data Sources Manager	When the sensor is deployed in the virtual infrastructure, the Orchestrator Sublayer sends a notification about the successful instantiation that is received by the Data Sources Manager component. The notification message contains all sensor instantiation information (sensor type, IP address, port, tenant ID, etc.).
3	Data Sources Manager	Virtual Sensors Descriptor	When a new virtual sensor is instantiated, the Data Sources Manager requests the available metadata (full set of attributes related to a specific sensor, including the output parameters, sensing period, sensor type, etc.) from the Virtual Sensors Descriptors component.
4	Data Sources Manager	Data Sources Instances	Once the Data Source Manager receives the instantiated virtual sensor metadata from the Virtual Sensors Descriptor component, it creates and configures a new data source instance (in this example is the SON Control Plane Data Source) to be prepared to start collecting (listening or polling) data from the sensor. Onboarded information (metrics types, events types, communication protocol, poll periods, etc.) and instantiation information (sensor type, IP address, port, tenant ID, etc.) are used by the Data Sources Manager to instantiate and configure the data source.
5	Data Sources Manager	Collector	The Data Sources Manager component provides the Collector component with the metrics and events that will be collected from the new virtual sensor, during the monitoring workflow.
6	Data Sources Manager	Raw Data Storage	Finally, the Data Sources Manager component provides to the Raw Data Storage component the new metrics (from the new virtual sensor) that should be stored and how (granularity of the stored metrics).

5.3. Sensors Instance Monitoring—Metrics and/or Events Workflow

After sensors are onboarded and instantiated in the virtual environment, data can start flowing, either pushed or polled, to the Monitoring & Discovery Framework. Figure 10 illustrates the sensors data flow (either metrics and/or events) towards the Monitoring & Discovery Framework.

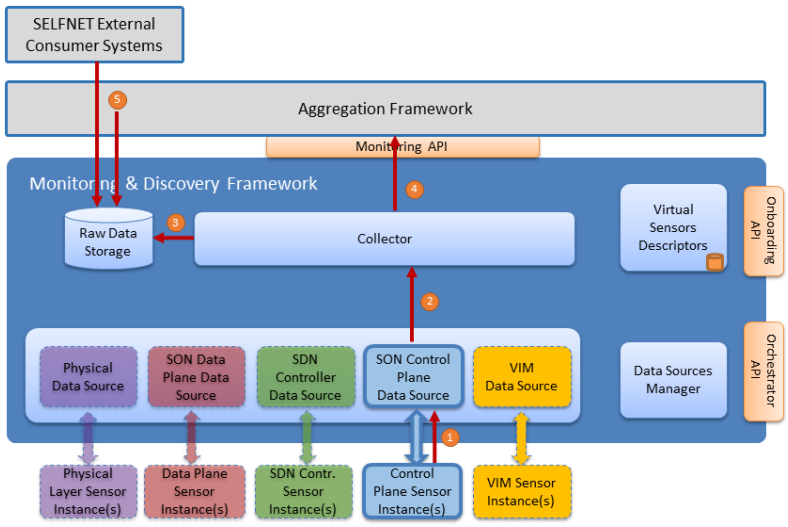


Figure 10. Sensor Monitoring—Metrics and/or Events Workflow.

The workflow steps are described in Table 7.

Table 7. Sensor Monitoring—Push Metrics and/or Events Workflow Steps.

Step	Source Component	Destination Component	Description
1	Sensor Instance	Data Source Instance	Data is provided by the sensor. Two alternative mechanisms are provided: Push and Polling.
2	Data Source Instance	Collector	The Data Source Instance receives the raw metrics and/or events and provides these to the Collector (specifically, to the Collector Message Bus) using a common API.
3	Collector	Raw Data Storage	The Collector component transforms/normalizes the received data and provides it to the Monitoring and Discovery Framework Raw Data Storage component.
4	Collector	Aggregation Framework	As in step 3, besides providing the received data to the Raw Data Storage component, the Collector component also delivers the collected data to the Aggregation Framework (using a message bus).
5	SELFNET external Consumer Systems	Raw Data Storage	Raw data (either metrics and/or events) can be retrieved/consumed by any SELFNET external system component using the exposed Raw Data Storage API.

5.4. Sensors Monitoring—Data Source Reconfiguration Workflow

During the sensor instance runtime, some specific parameters can be reconfigured without having to change the sensor instance. For example, one of the parameters that can be changed is the polling period. By default, the polling period is initially defined for each sensor type in the onboarded metadata. However, during the sensor runtime, this parameter can be changed (for example as an autonomic management decision) without affecting the sensor-running instance. This workflow is illustrated in Figure 11.

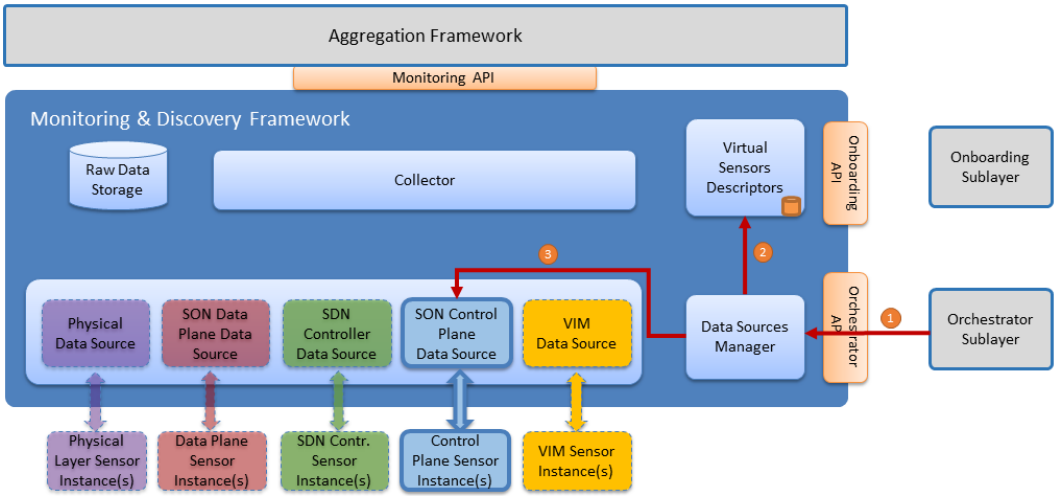


Figure 11. Sensor Monitoring—Data Source Reconfiguration Workflow.

The detailed steps of this workflow are described in Table 8.

Table 8. Sensor Monitoring—Poll Metrics Workflow Steps.

Step	Source Component	Destination Component	Description
1	Orchestrator Sublayer	Data Sources Manager	Orchestrator sublayer requests the Monitoring and Discovery Framework to modify the sensor instance polling period (e.g., due to an autonomic management decision).
2	Data Sources Manager	Virtual Sensors Descriptors	The Data Sources Manager requests the Virtual Sensors Descriptors for the sensor metadata information (if required).
3	Data Sources Manager	Data Source Instance	The Data Sources Manager requests the running data source instance to reconfigure the new polling period.

5.5. Sensors Instance Removal Workflow

Finally, when a sensor instance is removed from the infrastructure, the Data Source instance must be removed in the Monitoring and Discovery Framework. This workflow is illustrated in Figure 12.

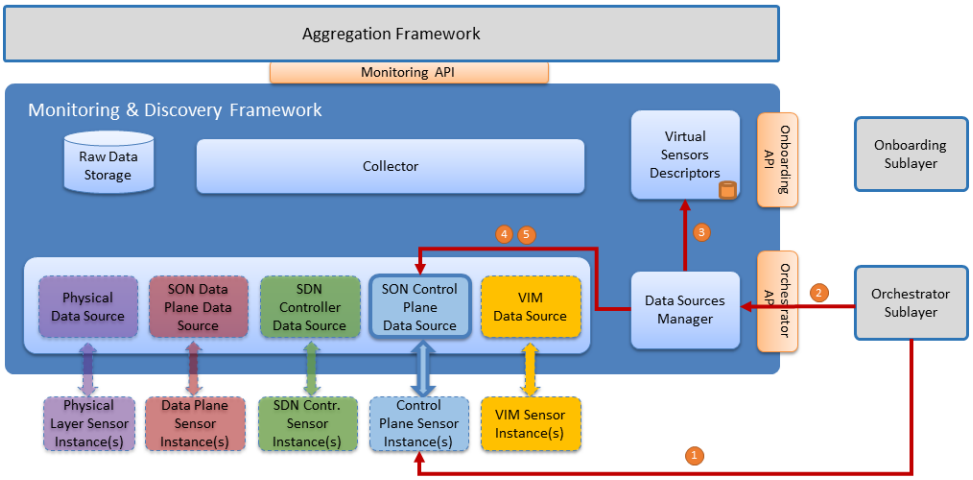


Figure 12. Sensor Instance Removal Workflow.

The workflow steps related with the removal of a sensor instance are described in Table 9.

Table 9. Sensor Instance Removal Workflow Steps.

Step	Source Component	Destination Component	Description
1	Orchestrator Sublayer	Sensor Instance	The Orchestrator Sublayer removes the running sensor instance (e.g., decision from the autonomic management system).
2	Orchestrator Sublayer	Data Sources Manager	The Orchestrator Sublayer notifies the Data Sources Manager that a specific sensor instance was removed.
3	Data Sources Manager	Virtual Sensors Descriptors	The Data Sources Manager requests the Virtual Sensors Descriptors for the sensor metadata information (if required).
4	Data Sources Manager	Data Source Instance	The Data Sources Manager requests the running data source instance to remove the configuration for the removed sensor instance.
5	Data Sources Manager	Data Source Instance	If the removed data sensor instance is the last instance of that data source type, the Data Sources Manager also removes the Data Sources Instance.

6. Implementation

The prototypal implementation of the Monitoring and Discovery framework has several challenges in order to fulfill the above described requirements. The gathered metrics have several sources depending on the level of collection (physical, virtual, sensor). For this purpose, different open sources have been tested for implementation. Firstly, the Common Monitoring Framework and how each component will interact in the monitoring process were described. Then, the implementation of Virtual Sensors Descriptors component, the implementation of each Data Source, the Data Sources Manager component, the Collector and the Raw Data Storage are detailed.

6.1. Common Monitoring Framework

In order to fulfill the Monitoring and Discovery requirements; we have chosen OpenStack project [39,40] and its Telemetry service as the baseline to the framework implementation process. OpenStack appears as a promised open source project to manage cloud platforms through a set of services (nova, neutron, keystone, telemetry, etc.), which could be integrated not only with traditional networks, but also with SDN and NFV approaches. Similarly, the Telemetry Project [41] was developed to facilitate the metering and monitoring of virtual resources from OpenStack deployments. This project manages different branches [42] in order to provide alarming service (Aodh), data collection service (Ceilometer/Monasca) [43] and time-series database and resource indexing service (Gnocchi) [44]. Ceilometer proposes an architecture based on plugins that allows easy scalability, extensibility, meter/alarm customization and tracking of available resources by means of the creation of new agents. A ceilometer agent collects not only information from OpenStack resources, such as Nova or Neutron [45], but also external sources, such as LibreNMS tool [46] and OpenDaylight (ODL) Time Series Data Repository (TSDR) [47]. The Ceilometer architecture is shown in Figure 13.

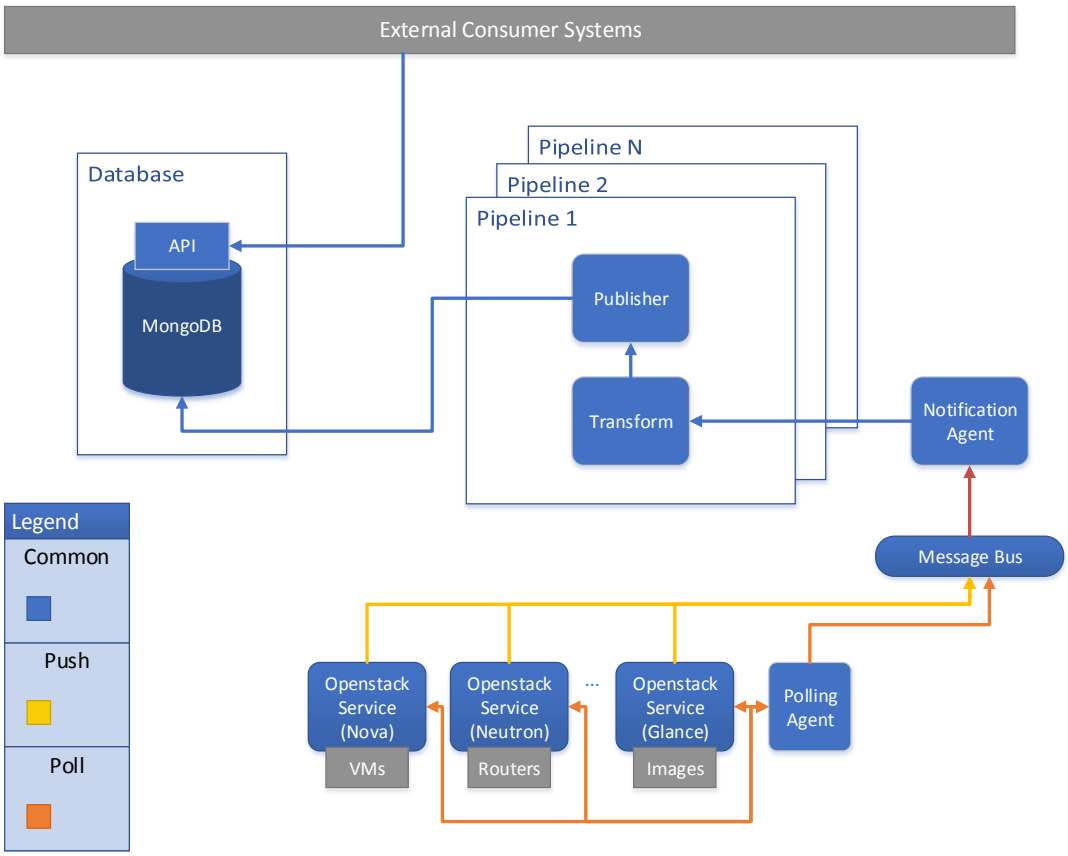


Figure 13. Ceilometer Architecture.

From a metric collector point of view, Ceilometer provides the polling and pushing methods for gathering data. The sources of information include not only OpenStack services, but also additional customizable metrics. However, the increase in the number of sources and resources can decrease exponentially the system performance and cause scalability problems. To address this issue, the Ceilosca project [48] combines Ceilometer (telemetry collector) and Monasca projects [49]. Ceilosca is used as a metric storage tool that provides a fast API for data access. This approach deals with the above mentioned scalability issue [50]. Figure 14 shows the mapping between a SELFNET Monitoring Framework with the Ceilosca approach and how it will operate.

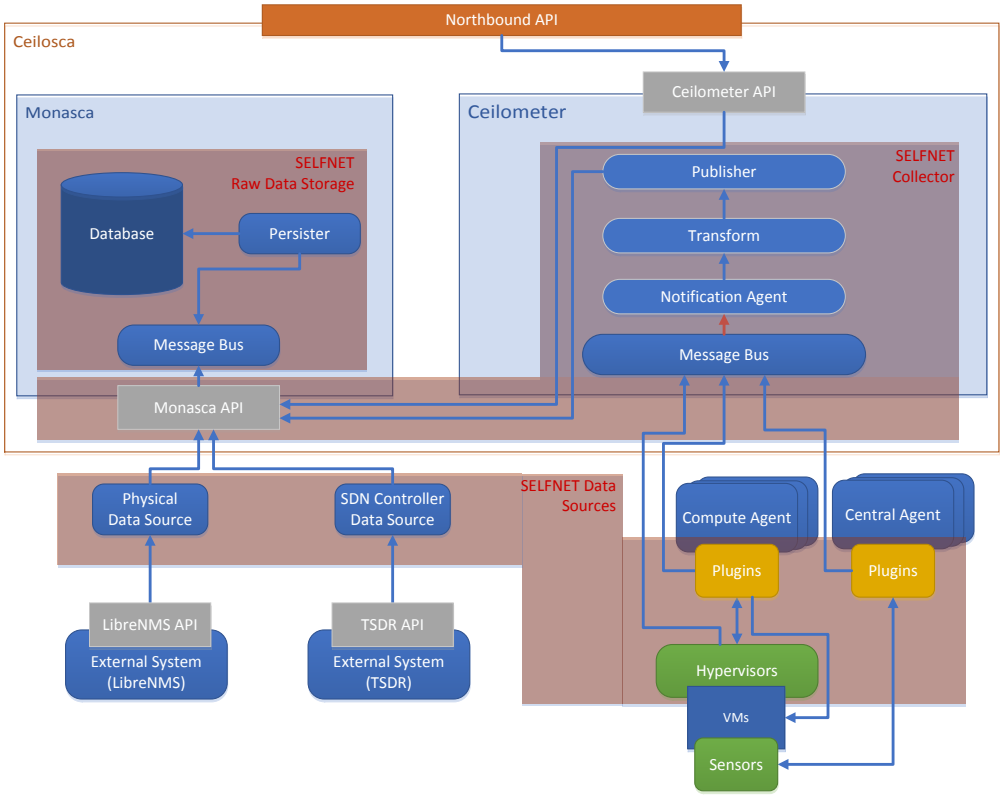


Figure 14. Mapping between Ceilosca and SELFNET Monitoring Framework.

The architecture has the following components:

- Data Sources: This component corresponds not only to traditional ceilometer data sources (OpenStack services) but also to new plugins (for sensors to be developed). Data sources are related to the data gathering task. This component is further detailed in Section 6.4.
- Collector: This component is composed of the Notification Agent and Ceilometer Pipeline and the Monasca API. This component is further detailed in Section 6.5.
- Raw Data Storage: This component storages meters and events. It is related to the Ceilosca database. This component is further detailed in Section 6.6.
- Northbound API: It represents the API exposed by the Monitoring and Discovery Framework. It is composed of the Ceilometer API.

6.2. *Virtual Sensors Descriptors*

Virtual Sensors Descriptors is responsible to receive, parse and store the information regarding onboarded sensor types available on SELFNET catalogue (Onborading Sublayer). For this purpose, it defines the next components:

- Message Queue client: to listen any action performed on the catalogue.
- Local Database: to store the sensor metrics and their kind of communication.
- REST server: Virtual Sensor Descriptors—Data Sources Manager interface.

The Monitor and Discovery framework listens for incoming messages provided by the Onboarding Sublayer when a change is made into the catalogue. The REST interface was developed using Python3 and a minimalist Web micro framework named Falcon. Since the sensor metadata is not relational, and it will only exhibit onboarded sensor data, the local database is a MongoDB database, a NoSQL database system. The interface with the VNF Onboarding Sublayer was developed also using Python3 and RabbitMQ, as the Message Bus.

The Virtual Sensors Descriptors expose a REST interface to the Data Sources Manager allowing it to reach the catalogue of sensor information. This interface needs to disclose a single endpoint to retrieve information. The client needs to provide the sensor type identifier in order to receive a JSON response with the information available on the local sensor catalogue. The Message Queue client is prepared to receive three types of actions:

- Create—Add a new monitoring piece of metadata to the local database.
- Update—Replace the monitoring metadata.
- Delete—Delete the stored metadata for a specific sensor.

Each collected metric is composed of a unique name, "metric-name" and a set of sample metrics, each one identified by a name, a resource which provides the origin of the value, a unit and a type (e.g., cumulative or delta). The Virtual Sensor Descriptors also exposes the sensor communication to the Data Sources Manager, composed of the protocol, a set of endpoints with the metric id and the endpoint value, the method (push or poll) and a list with metadata composed of a key value pair.

6.3. Data Sources Manager

To create the corresponding Data Sources in line with the sensor specifications (delivered data, protocol and period), the Data Sources Manager component is implemented. It dynamically creates specific Data Sources to map the type of sensor instantiated in the infrastructure, establishing the communication channel to gather information. In order to receive information about sensor activities (instantiation, reconfiguration, delete), a REST API interface between the Data Sources Manager and the Orchestrator is implemented. Then, the information must be classified by the Action Selector component, as is illustrated in Figure 15. Action Selector is used to analyze the type of action performed by the Orchestrator through specific messages, identifying which operation is done (sensor deployment, remove or reconfiguration). According to the message obtained from the orchestrator, the action selector decides which operation is done and notifies the respective module.

Three functional blocks are implemented: (i) the sensor deployment manager; (ii) the sensor remove manager; and (iii) sensor re-configuration manager. Furthermore, an interface is implemented between Data Sources Manager and Virtual Sensors Descriptors component, which can retrieve the necessary information, such as the type and ID of SDN/NFV applications, the metrics collected by the sensor, the data transmission mode (poll/push), among others. The retrieved information is temporarily stored in the Cache, called Sensor Description Cache. The sensor description is used to manipulate the sensor during the sensor lifecycle.

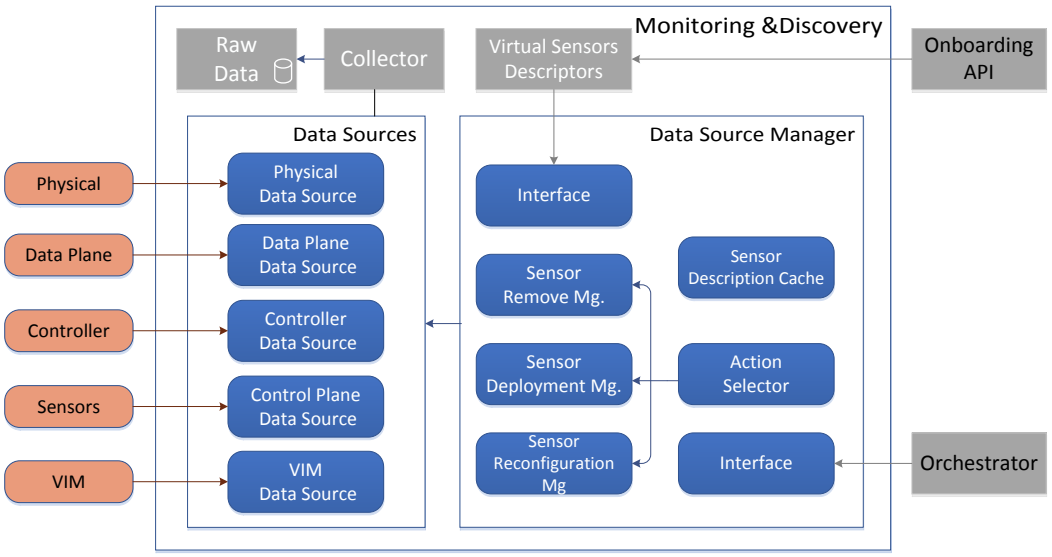


Figure 15. Data Source Manager: Sensor Operations.

6.4. Data Sources Instances

A Data Source is defined as an interface between the Monitoring Framework and the monitored element and is able to gather data from the corresponding monitored device. For this purpose, the SELFNET framework takes advantage of several open source monitoring tools, as described in Table 10.

Table 10. Open Source dependences.

Module	Tool
Physical Infrastructure	LibreNMS [46]
SDN Controller	ODL [51]
SON Data Plane	ODL-TSDR [47]
VIM	OpenStack [40]
Control Plane Sensor	Ceilometer [41]

6.5. Collector

Having in mind the mapping between SELFNET architecture and Ceilosca (Section 6.1). The Collector is composed of the Monasca API and four Ceilometer components: the Message Bus, the Notification Agent, the Transform and the Publisher. The Notification Agent consumes data from the Message Bus, delivering it to the Transform, where the data is normalized. Then the Publisher(s) publish the data into the destination: the Ceilosca database (by using the Monasca API) and external consumer systems (such as a message bus used by the Aggregation Framework). It is important to note that the Transform has more capabilities as is mentioned in Section 6.1. Moreover, the Monasca API can be used to directly publish telemetry data from non-OpenStack services (e.g., LibreNMS, ODL-TSDR) without using Ceilometer.

6.6. Raw Data Storage

Ceilometer provides several ways to store both meters and events. By default, Ceilometer stores meters and events in a MongoDB, a NoSQL database, where every record is stored in the database (raw data). As of the date, the Ceilometer project recommends to replace MongoDB by the TDBaaS Gnocchi (Time Series Database as a Service) to store meters, while keeping MongoDB for events storage, especially when facing low latency use cases. The main reason for this change is the scalability problems of MongoDB when stored data are queried (bad performance when increase the amount of

records stored). SELFNET Monitoring Framework presents some alternatives to store the data. On one hand, Monasca, by default, stores data using InfluxDB (TDBaaS like Gnocchi). On the other hand, Ceilosca provides a faster and more complete API that enables, not only operations like aggregation, max, count, avg, over data; but also a way to insert non-telemetry data (external to Ceilometer).

6.7. Validation–Monitoring GUI

The Monitoring and Discovery Framework has been implemented in a virtual environment as a proof of concept, over a single physical server (ThinkServer TD350: Intel® Xeon® Serie E5-2600 v3 -16 cores, DDR4 512 GB—2133 MHz). Six virtual machines have been deployed through VirtualBox hypervisor, as is shown in Figure 16. These VMs represent the Controller Node, Compute Node, Telemetry Node, OpenDaylight Node, LibreNMS Node and the Monitoring Server.

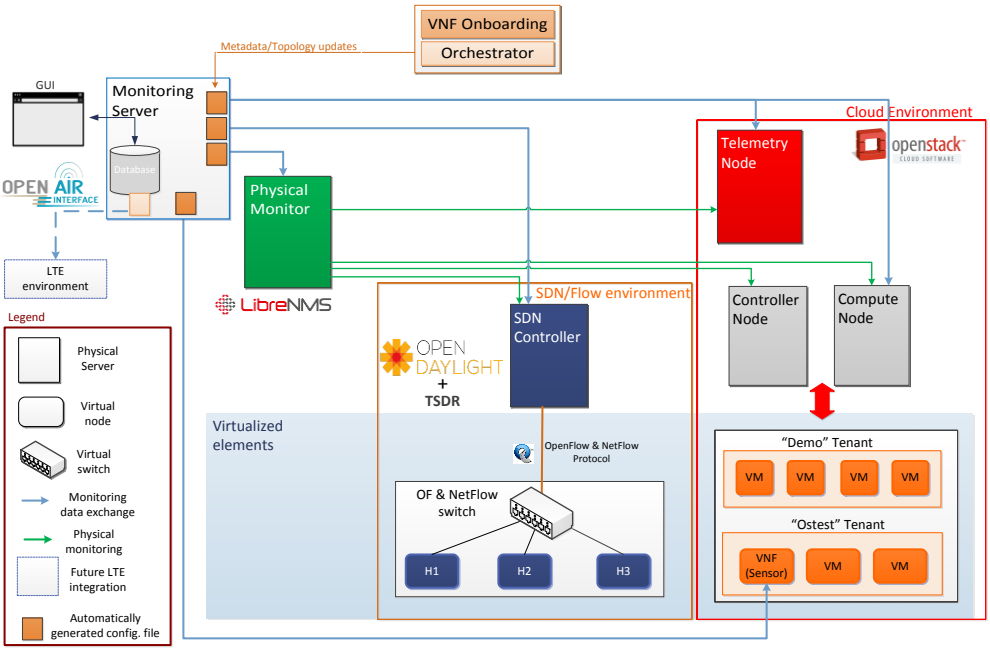


Figure 16. SELFNET Monitoring Framework Testbed.

The Monitoring Server retrieves the required information from data sources: Physical Monitor (LibreNMS), Cloud Environment (OpenStack), SDN/Flow environment (ODL+TSDR), LTE environment (OpenAir) and Sensors. The Monitoring Server manages the communication not only with the VNF Onboarding Sublayer and the Orchestrator Sublayer, but also with all data sources through a set of configuration files. Physical Monitor is able to collect data from SDN Controller, Compute Node, Telemetry and Controller Node, which emulates a Physical Server. In turn, Telemetry Node collects information from each OpenStack service available in the cloud environment. For its part, Physical Monitor retrieves sensor metrics through the Sensor API. Regarding to the collection of LTE metrics, this is part of the ongoing work. Moreover, the communication between these nodes, are done through two networks, as follows:

(1) The Management Network (MGNT) is in charge of the communication between OpenStack, LibreNMS and ODL Servers.

(2) The Public Network provides internet access and must be reachable by anybody. In particular, some OpenStack API (Neutron, Horizon) are exposed via this network.

Furthermore, two OpenStack services are mainly involved in the communication between virtual instances (VMs); (i) the Nova Service that enables the instantiation of a new virtual machine and (ii) the Neutron Service that is responsible of the management of the virtual networks.

In order to visualize the collected information, a simple Monitoring GUI was developed as a three-tier architecture based on a client-server approach. This GUI shows selected data from physical layer devices, virtual layer devices, LTE, sensors and flows level (Figure 17).

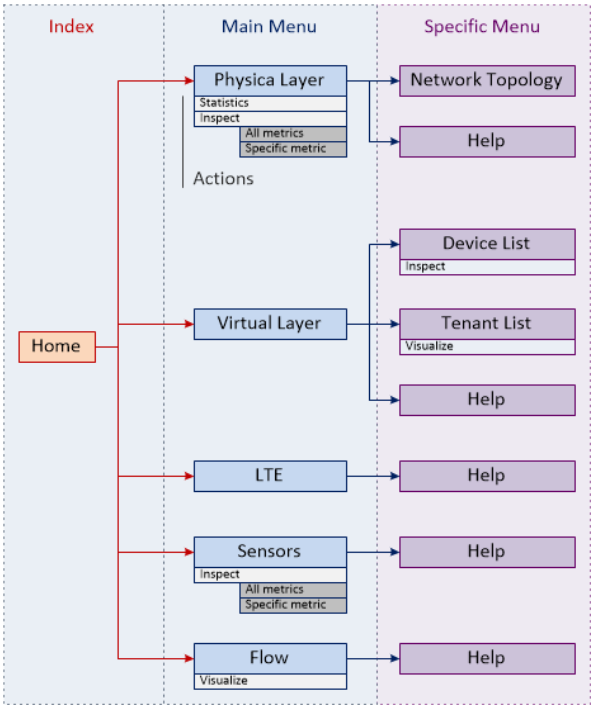
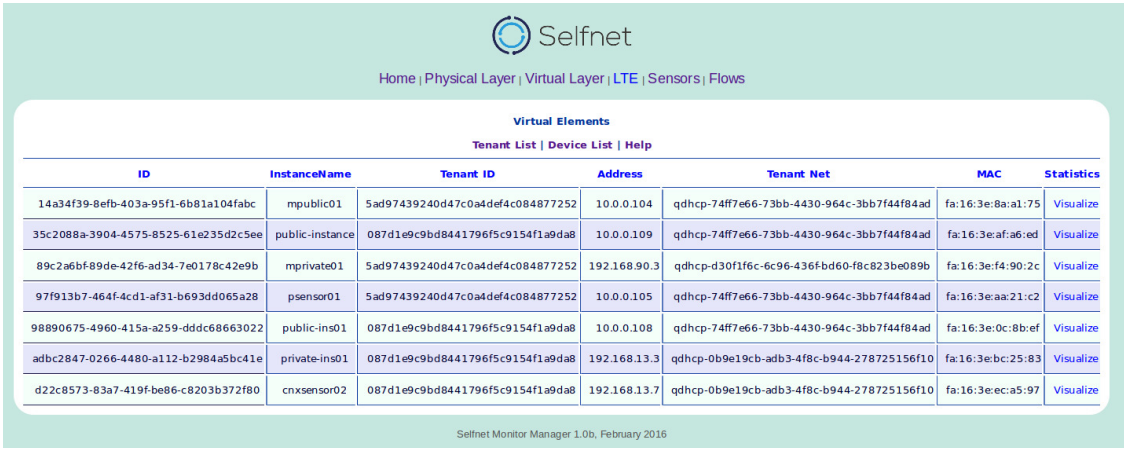


Figure 17. SELFNET Monitoring GUI navigability.

For example, the view of the physical layer displays a list of all discovered network devices and their most relevant parameters: device identification, hostname, location, port and MAC address. The Virtual Layer View shows the tenant list (List of tenant with their respective ID, name and description) and the device list (related information of each virtual instance) as is illustrated in Figure 18.



ID	InstanceName	Tenant ID	Address	Tenant Net	MAC	Statistics
14a34f39-8efb-403a-95f1-6b81a104fabc	mpublic01	5ad97439240d47c0a4def4c084877252	10.0.0.104	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:8a:a1:75	Visualize
35c2088a-3904-4575-8525-61e235d2c5ee	public-instance	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.109	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:af:a6:ed	Visualize
89c2a6bf-89de-42f6-ad34-7e0178c42e9b	mprivate01	5ad97439240d47c0a4def4c084877252	192.168.90.3	qdhcp-d30f1f6c-6c96-436f-bd60-f8c823be089b	fa:16:3e:f4:90:2c	Visualize
97f913b7-464f-4cd1-af31-b693dd065a28	psensor01	5ad97439240d47c0a4def4c084877252	10.0.0.105	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:aa:21:c2	Visualize
98890675-4960-415a-a259-dddc68663022	public-ins01	087d1e9c9bd8441796f5c9154f1a9da8	10.0.0.108	qdhcp-74ff7e66-73bb-4430-964c-3bb7f44f84ad	fa:16:3e:0c:8b:ef	Visualize
adbc2847-0266-4480-a112-b2984a5bc41e	private-ins01	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.3	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:bc:25:83	Visualize
d22c8573-83a7-419f-be86-c8203b372f80	cnxsensor02	087d1e9c9bd8441796f5c9154f1a9da8	192.168.13.7	qdhcp-0b9e19cb-adb3-4f8c-b944-278725156f10	fa:16:3e:ec:a5:97	Visualize

Figure 18. List of SELFNET virtual elements on the Virtual Layer View.

For its part, the Flow View describes information about the gathered flow metrics (OpenFlow and NetFlow) and the Sensor View displays a list of all deployed sensors and their most relevant parameters: sensor identification, IP Address, tenant identification, tenant network identification, location, type of sensor, and their values and statistics; as is illustrated in Figure 19.

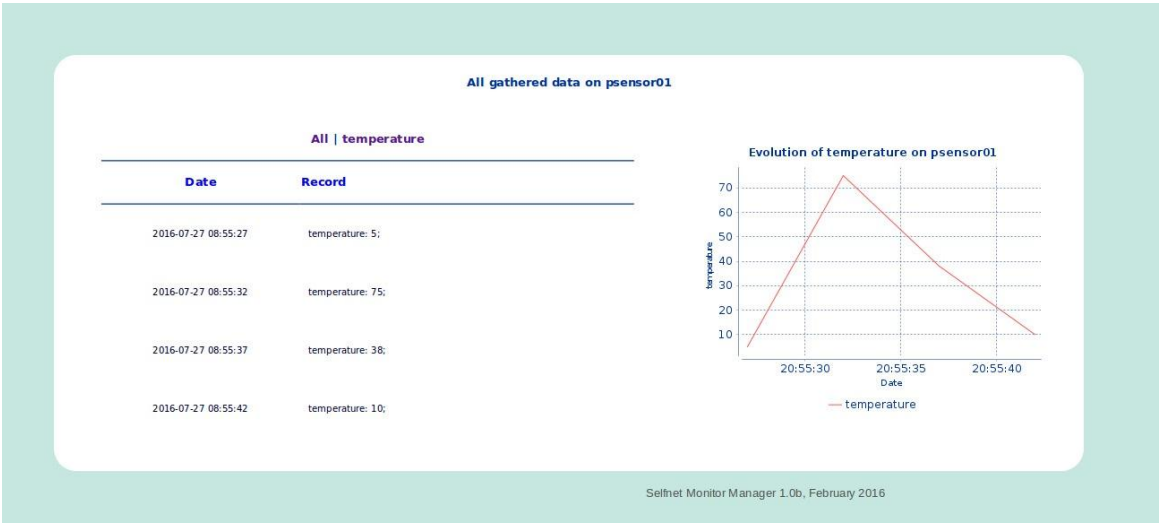


Figure 19. Details of information gathered by a SELFNET Sensor.

It is important to note that "psensor01" is a simple temperature sensor. It is running in a virtual machine of a specific tenant (ostest), from which the data is retrieved every 5 s. The monitored metrics are defined in the Onboarding Sublayer and the instantiation information in the Orchestrator Sublayer. This example shows the temperature values for each sample and their evolution over time. This module is able to (i) customize the monitored metrics by each sensor type and (ii) gather these metrics from the respective data source.

An additional way in validating the capabilities of the framework includes the analysis of specific Use Cases. The SELFNET Self-Optimization Use Case [52] improves the end users perceived QoE in video streaming services. In this case, the sensors are deployed in the virtualized infrastructure. Sensors monitor network state metrics, new ultrahigh definition (U-HD) video and energy related metrics. The combination of these low level metrics is combined to address innovative Health of Network (HoN) composite metrics. SELFNET is able to detect and respond when QoE levels either have fallen below or are predicted to fall below expected levels. Similarly, the energy efficiency of the system can be analyzed to proactively managing the energy use of resources across the network. Meanwhile, the SELFNET Self-Healing Use Case [53] is applied to reactively or preventively deal with the detected or predicted network failures. The different metrics collected by the sensors give a global view of the available networking resources and services running on virtualized infrastructure. In this way, the system can be adapted to new load situations through the instantiation of virtual load balancers in strategical points. Additionally, SELFNET is able to use the historical references to predict high-resource demands before they happen and take actions about them.

7. Conclusions and Future Work

The design and prototype implementation of the SELFNET Monitoring and Discovery framework were presented in this document. The proposed architecture is aligned with the latest and innovative trends at the ongoing network monitoring research activities and guarantee easy deployment and scalability. The SELFNET Monitoring and Discovery Framework has included the achieved innovations towards monitoring the different sources within 5G Infrastructures. Particularly, the SELFNET Monitoring and Discovery framework will be able to collect and organize monitored information according to different criteria, such as virtual instances by tenant, metrics by virtual instances or physical devices by network location. Moreover, the SELFNET Monitoring facilitates not only to query the gathered information, but also to manage large amounts of data from heterogeneous data sources. The information of each SELFNET layer may be correlated to provide enhanced information of the

network status such as the relationship between virtual instances and their respective physical device, the physical and virtual instances where a sensor is running. This information will be available to SELFNET Aggregation and Correlation, and hence to the SELFNET Analysis process.

The future works include the development of the other SELFNET sublayers, such as SON Control Plane (Sensors), Analyzer, Autonomic Management, Orchestrator, VNF Onboarding, among others. In this way, the evaluation of the use cases and the performance metrics definition will be performed. Similarly, the use of correlation and aggregation techniques in order to define new global HoN metrics capable of identifying network failures, risk and the corresponding responses to mitigate the network problems is included. In the same way, the use of the monitoring available information to predict the future network behaviours in order to prevent the QoS/QoE degradation is also an interest topic. For instance, the potential network congestion and service failures in a HD video application can be mitigated through the prediction of an increase of network traffic. The system can automatically install NFV virtual routers to perform dynamic traffic balancing to adapt the operational capacity with the time-varying traffic demands.

Acknowledgments: This work was funded by the European Commission Horizon 2020 5G-PPP Programme under Grant Agreement number H2020-ICT-2014-2/671672-SELFNET (Framework for Self-Organized Network Management in Virtualized and Software-Defined Networks). Ángel Leonardo Valdivieso Caraguay is supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program N. 2543-2012.

Author Contributions: Á.L. Valdivieso Caraguay and L.J. García Villalba are the authors who mainly contributed to this research, performing experiments, analysis of the data and writing the manuscript. All authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BSS	Business Support Systems
DPI	Data Packet Inspection
ETSI	European Telecommunications Standards Institute
HoN	Health of Network
LTE	Long Term Evolution
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NOS	Network Operating System
ODL	OpenDaylight
OSS	Operational Support Systems
QoS	Quality of Service
QoE	Quality of Experience
SDN	Software Defined Networking
SELFNET	Self-Organized Network Management in Virtualized and Software Defined Networks
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SON	Self Organizing Networks
TSDB	Time Series Database
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VN	Virtual Network

References

1. Barona, L.; Valdivieso, A.; Sotelo, M.; García, L. Key Technologies in the Context of Future Networks: Operational and Management Requirements. *Futur. Internet* **2017**, *9*, 1–17. [CrossRef]
2. EU SELFNET Project—Self-Organized Network Management in Virtualized and Software Defined Networks. Project Reference: H2020-ICT-2014-2/671672. Funded under: H2020. Available online: http://www.selfnet-5g.eu (accessed on 30 March 2017).
3. Feamster, N.; Rexford, J.; Zegura, E. The Road to SDN: An Intellectual History of Programmable Networks. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–98. [CrossRef]
4. Hu, F.; Hao, Q.; Bao, K. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1–27. [CrossRef]
5. Open Networking Foundation (ONF). Available online: https://www.opennetworking.org/sdn-resources/sdn-definition (accessed on 30 March 2017).
6. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
7. Jain, R.; Paul, S. Network Virtualization and Software Defined Networking for Cloud Computing: A survey. *IEEE Commun. Mag.* **2013**, *51*, 24–31. [CrossRef]
8. Mijumbi, R.; Serrat, J.; Gorricho, J.L.; Bouten, N.; Turck, F.; Boutaba, R. Network Function Virtualization: State-of-the-art and Research Challenges. *IEEE Commun. Surv. Tutor.* **2015**, *18*, 236–262. [CrossRef]
9. Garay, J.; Unzilla, J.; Jacob, E. Service Description in the NFV Revolution: Trends, Challenges and a Way Forward. *IEEE Commun. Mag.* **2016**, *54*, 68–74. [CrossRef]
10. ETSI Industry Specification Group (ISG). *Network Function Virtualization (NFV) Architectural Framework*; ETSI: Sophia Antipolis, France, 2013; pp. 1–21.
11. EU CROWD Project. Connectivity Management for Energy Optimised Wireless Dense Networks. Project Reference: 318115. Funded under: FP7-ICT. Available online: http://www.ict-crowd.eu/ (accessed on 30 March 2017).
12. EU 5G-NORMA Project. 5G Novel Radio Multiservice Adaptive Network Architecture. Project Reference: 671584. Funded under: H2020-ICT-2014-2. Available online: https://5gnorma.5g-ppp.eu/ (accessed on 30 March 2017).
13. EU MCN Project. Mobile Cloud Networking. Project Reference: 318109. Funded under: FP7. Available online: http://www.mobile-cloud-networking.eu/site/ (accessed on 30 March 2017).
14. EU UNIFY Project. Unifying Cloud and Carrier Networks. Project Reference: 619609. Funded under: FP7. Available online: https://www.fp7-unify.eu/ (accessed on 30 March 2017).
15. EU T-NOVA Project. Network Functions as-a-Service over Virtualised Infrastructures. Project Reference: 619520. Funded under: FP7. Available online: http://www.t-nova.eu/ (accessed on 30 March 2017).
16. Yang, Y.; Cheng, W.; Yang, C.; Chen, S.; Jian, F. The Implementation of Real-Time Network Traffic Monitoring Service with Network Functions Virtualization. In Proceedings of the 2015 IEEE International Conference on Cloud Computing and Big Data (CCBD), Taipei, Taiwan, 4–6 November 2015; pp. 279–286.
17. Liu, G.; Trotter, M.; Ren, Y.; Wood, T. NetAlytics: Cloud-Scale Application Performance Monitoring with SDN and NFV. In Proceedings of the 17th ACM International Middleware Conference, Trento, Italy, 12–16 December 2016; pp. 440–445.
18. Gardikis, G.; Koutras, I.; Mavroudis, G.; Costicoglou, S.; Xilouris, G.; Sakkas, C.; Kourtis, A. An Integrating Framework for Efficient NFV monitoring. In Proceedings of the IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Korea, 6–10 June 2016; pp. 1–5.
19. Shu, Z.; Wan, J.; Lin, J.; Wang, S.; Li, D.; Rho, S.; Yang, C. Traffic Engineering in Software-defined Networking: Measurement and Management. *IEEE Access* **2016**, *4*, 3246–3256. [CrossRef]
20. Neves, P.; Calé, R.; Costa, M.R.; Parada, C.; Parreira, B.; Alcaraz-Calero, J.; Schotten, H. The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm. *Int. J. Distrib. Sens. Netw.* **2016**, *2*, 1–17. [CrossRef]
21. ETSI Mobile Edge Computing (MEC). *Mobile-Edge Computing: Introductory Technical White Paper*; ETSI: Sophia Antipolis, France, 2014; pp. 1–36.

22. Amit, N.; Gordon, A.; Har'El, N.; Ben-Yehuda, M.; Landau, A.; Schuster, A.; Tsafrir, D. Bare-metal Performance for Virtual Machines with Exitless Interrupts. *Commun. ACM* **2015**, *59*, 108–116. [CrossRef]
23. Muñoz, R.; Vilalta, R.; Casellas, R.; Martinez, R.; Szyrkowiec, T.; Autenrieth, A.; López, D. Integrated SDN/NFV Management and Orchestration Architecture for Dynamic Deployment of Virtual SDN Control Instances for Virtual Tenant Networks. *J. Opt. Commun. Netw.* **2015**, *7*, B62–B70. [CrossRef]
24. ETSI Industry Specification Group (ISG). *ETSI NFV Management and Orchestration (ETSI MANO)*; ETSI: Sophia Antipolis, France, 2014; pp. 1–184.
25. Ljung, L. Analysis of Recursive Stochastic Algorithms. *IEEE Trans. Autom. Control* **1977**, *22*, 551–575. [CrossRef]
26. Ali, M.; Khompatraporn, C.; Zabinsky, Z. A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems. *J. Glob. Optim.* **2005**, *31*, 635–672. [CrossRef]
27. Vila, L. A Survey on Temporal Reasoning in Artificial Intelligence. *AI Commun.* **1994**, *7*, 4–28.
28. Garvey, A.; Lesser, V. A Survey of Research in Deliberative Real-time Artificial Intelligence. *Real-Time Syst.* **1994**, *6*, 317–347. [CrossRef]
29. Mitra, S.; Pal, S.; Mitra, P. Data Mining in Soft Computing Framework: A Survey. *IEEE Trans. Neural Netw.* **2002**, *13*, 3–14. [CrossRef] [PubMed]
30. SELFNET Consortium. *Deliverable 2.1: Use Cases Definition and Requirements of the System and Its Components*; Eurescom: Heidelberg, Germany, 2015.
31. SELFNET Consortium. *Deliverable 5.3: Report and Prototypical Implementation of the Integration of the Algorithms and Techniques Used to Provide Intelligence to the Decision-Making Framework*; Eurescom: Heidelberg, Germany, 2016.
32. Martin-Flatin, J. Push vs. Pull in Web-based Network Management. In Proceedings of the Sixth IFIP/IEEE International Symposium on Distributed Management for the Networked Millennium, Boston, MA, USA, 24–28 May 1999; pp. 3–18.
33. Li, N.; Du, Y.; Chen, G. Survey of Cloud Messaging Push Notification Service. In Proceedings of the 2013 International Conference on Information Science and Cloud Computing Companion (ISCC-C), 7–8 December 2013; pp. 273–279.
34. Case, J.; Fedor, M.; Schoffstall, M.; Davin, J. Simple Network Management Protocol (SNMP)—RFC 1157. Available online: http://www.rfc-editor.org/info/rfc1157 (accessed on 31 March 2017).
35. Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. Hypertext Transfer Protocol–HTTP/1.1 - RFC 2616. Available online: https://www.rfc-editor.org/info/rfc2616 (accessed on 30 March 2017).
36. Google Developers. Google Cloud Messaging for Android. Available online: http://developers.google.com/cloud-messaging/ (accessed on 30 March 2017).
37. Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence—RFC 3921. Available online: https://www.rfc-editor.org/info/rfc3921 (accessed on 31March 2017).
38. SELFNET Consortium. *Deliverable 3.1: Report and Prototype Implementation of the NFV & SDN Repository*; Eurescom: Heidelberg, Germany, 2016.
39. Sefraoui, O.; Aissaoui, M.; Eleuldj, M. OpenStack: Toward an Open-source Solution for Cloud Computing. *Int. J. Comput. Appl.* **2012**, *55*, 38–42. [CrossRef]
40. OpenStack Project. Available online: https://www.openstack.org/ (accessed on 30 March 2017).
41. Telemetry. Available online: http://docs.openstack.org/admin-guide/telemetry.html (accessed on 30 March 2017).
42. Telemetry Project. Available online: https://wiki.openstack.org/wiki/Telemetry (accessed on 30 March 2017).
43. Ceilometer Project. Available online: http://docs.openstack.org/developer/ceilometer/index.html (accessed on 30 March 2017).
44. Gnocchi. Available online: https://wiki.openstack.org/wiki/Gnocchi (accessed on 30 March 2017).
45. Ceilometer Telemetry Metrics. Available online: http://docs.openstack.org/admin-guide/telemetry-measurements.html (accessed on 30 March 2017).
46. LibreNMS. Available online: http://www.librenms.org/ (accessed on 30 March 2017).
47. OpenDaylight TSDR. Available online: https://wiki.opendaylight.org/view/Project_Proposals:Time_Series_Data_Repository (accessed on 30 March 2017).

48. Ceilosca Project. Available online: https://www.openstack.org/summit/tokyo-2015/videos/presentation/ceilometer-monascaceilosca (accessed on 30 March 2017).
49. Monasca. Available online: https://wiki.openstack.org/wiki/Monasca (accessed on 30 March 2017).
50. Ceilosca. Available online: https://wiki.openstack.org/wiki/Ceilosca (accessed on 30 March 2017).
51. Opendaylight SDN Controller. Available online: https://www.opendaylight.org/software/downloads/boron-sr2 (accessed on 30 March 2017).
52. Nightingale, J.; Wang, Q.; Calero, J.; Chirivella, E.; Ulbricht, M.; Alonso, J.; Reinsch, C. QoE-driven, Energy-aware Video Adaptation in 5G networks: The SELFNET Self-optimisation Use Case. *Int. J. Distrib. Sens. Netw.* **2016**, *12*, 1–15. [CrossRef]
53. Santos, J.; Alheiro, R.; Andrade, L.; Valdivieso, L.; García, L.; Alcaraz, J. SELFNET Framework self-healing capabilities for 5G mobile networks. *Trans. Emerg. Telecommun. Technol.* **2016**, *27*, 1225–1232. [CrossRef]